Fundamentals of Designing a Data Warehouse

Sensible techniques for developing a data warehousing environment which is relevant, agile, and extensible

Melissa Coates
BI Architect, SentryOne
sentryone.com

Microsoft® MVP Most Valuable Professional

SQL Chick

Blog: sqlchick.com
Twitter: @sqlchick

Presentation content last updated: 2/15/2017

Fundamentals of Designing a Data Warehouse

Agenda

1. Overview of the Need for Data Warehousing
2. DW Design Principles
3. Dimension Design
4. Fact Design
5. When to Use Columnstore or Partitioning
6. DW Tips
7. SSDT 'Database Project' Tips
8. Planning Future Growth of the DW

All syntax shown is from SQL Server 2016.

Screen shots are from SQL Server Data Tools in Visual Studio 2015.

Fundamentals of Designing a Data Warehouse

Out of Scope

- ✓ ETL patterns and techniques
- ✓ Source control
- ✓ Deployment practices
- ✓ Master data management
- ✓ Data quality techniques

- ✓ Semantic layer, OLAP, cubes
- ✓ Front-end reporting
- ✓ Security
- ✓ Tuning & monitoring
- ✓ Automation techniques

Overview of the
Need for Data Warehousing

First Let's Get This Straight...

Data Warehousing is not dead!

Data warehousing can be "uncool" but it doesn't have to be if you adopt modern data warehousing concepts & technologies such as:

- ✓ Data lake
- ✓ Hadoop
- ✓ Real-time
- ✓ Large data volume
- ✓ Data virtualization
- ✓ Hybrid & cloud
- ✓ Automation
- ✓ Bimodal environments

Transaction System vs. Data Warehouse

OLTP	Data Warehouse

Goal:
- ✓ Operational transactions
- ✓ "Writes"

Scope:
One database system

Example Objectives:
- ✓ Process a customer order
- ✓ Generate an invoice

Goal:
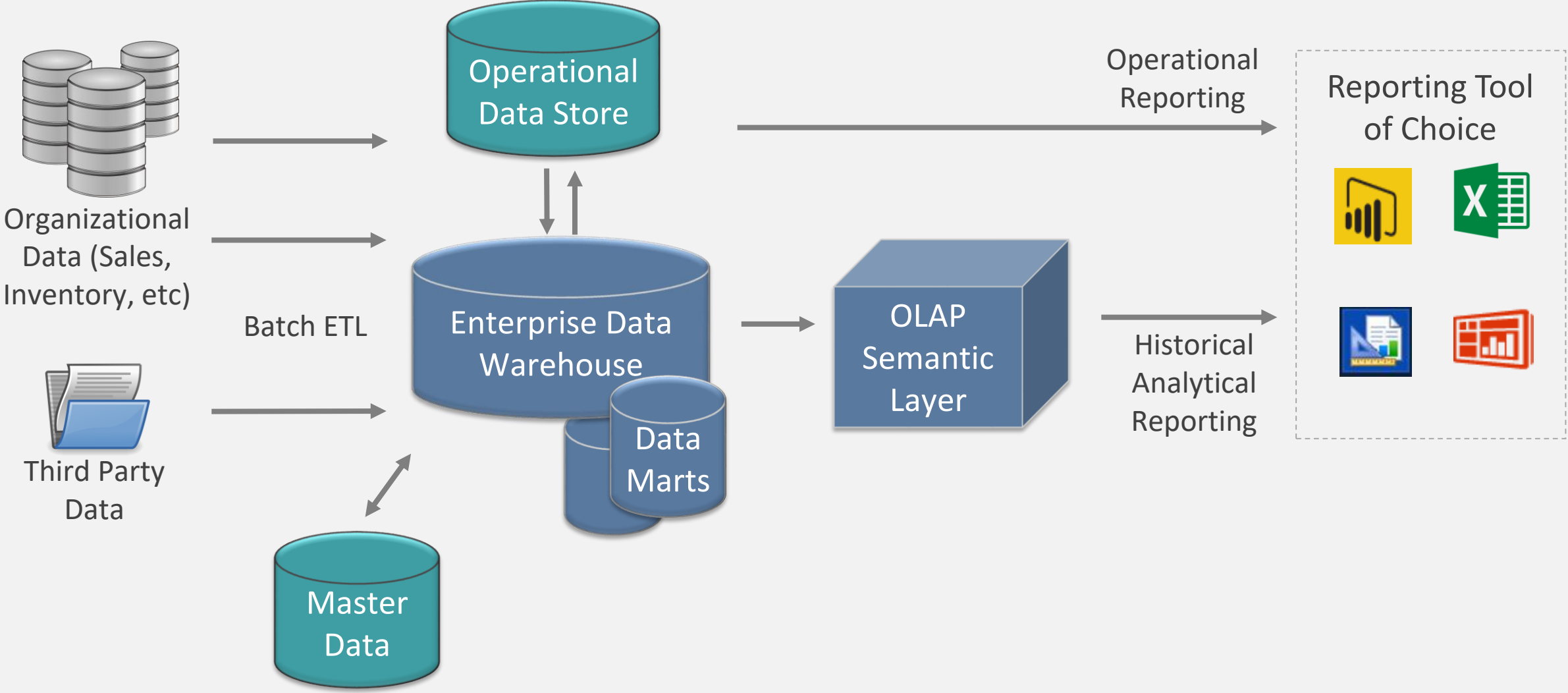- ✓ Informational and analytical
- ✓ "Reads"

Scope:
Integrate data from multiple systems

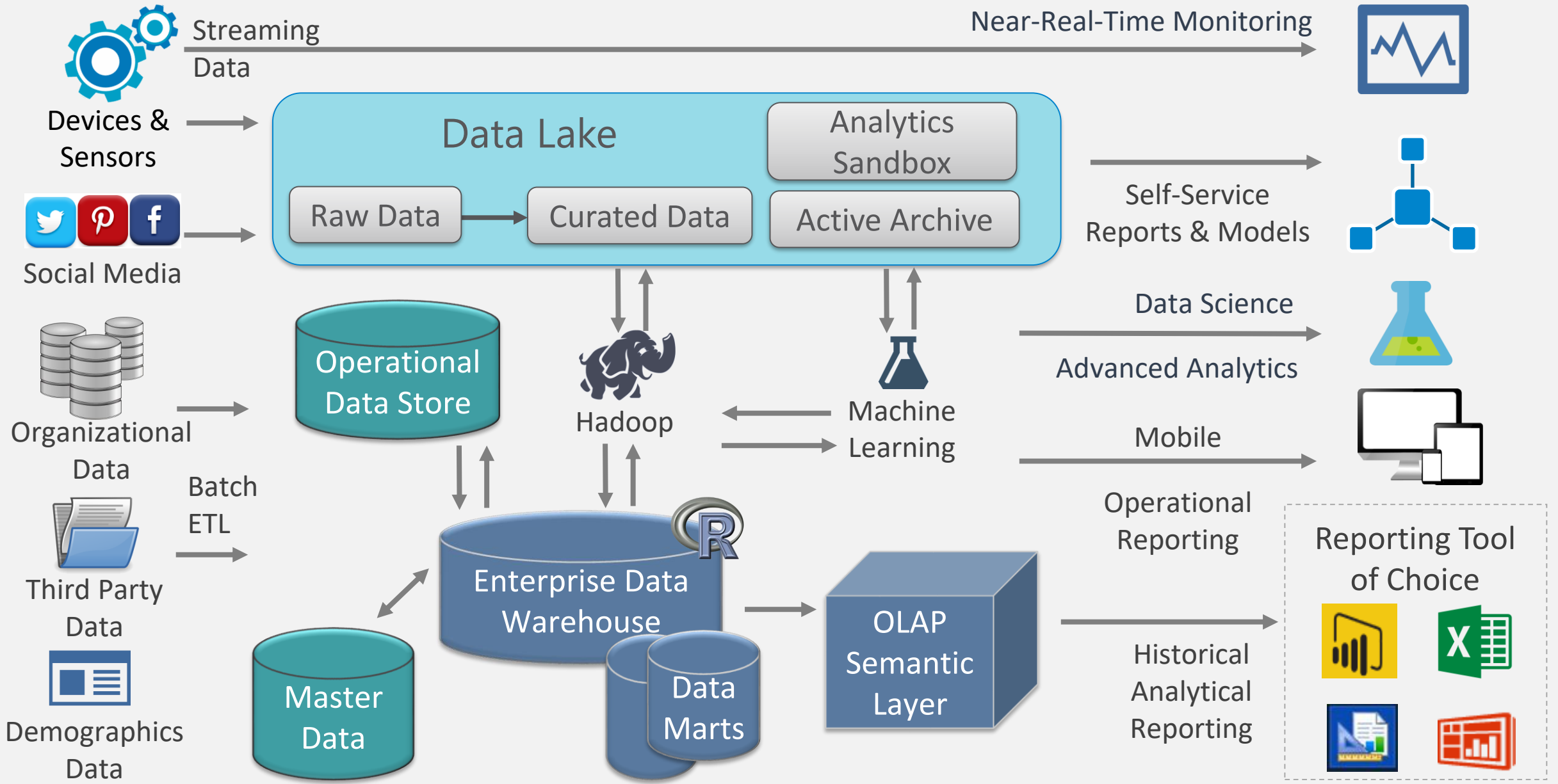**Example Objectives:**
- ✓ Identify lowest-selling products
- ✓ Analyze margin per customer

DW+BI Systems Used to Be Fairly Straightforward
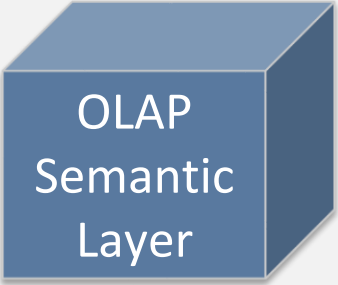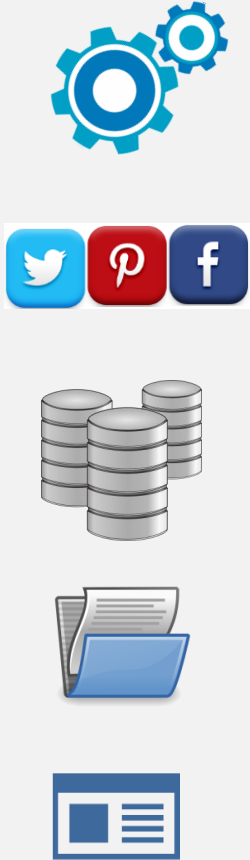
DW+BI Systems Have Grown in Complexity

Streaming Data

Near-Real-Time Monitoring

Devices & Sensors

Social Media

Data Lake

Analytics Sandbox

Raw Data → Curated Data

Active Archive

Self-Service Reports & Models

Organizational Data

Operational Data Store

Hadoop

Machine Learning

Data Science

Advanced Analytics

Mobile

Batch ETL

Third Party Data

Master Data

Enterprise Data Warehouse

Data Marts

OLAP Semantic Layer

Operational Reporting

Historical Analytical Reporting

Reporting Tool of Choice

Demographics Data

Data Warehouse Design Principles

3 Primary Architectural Areas

Integrate Data from Multiple Sources

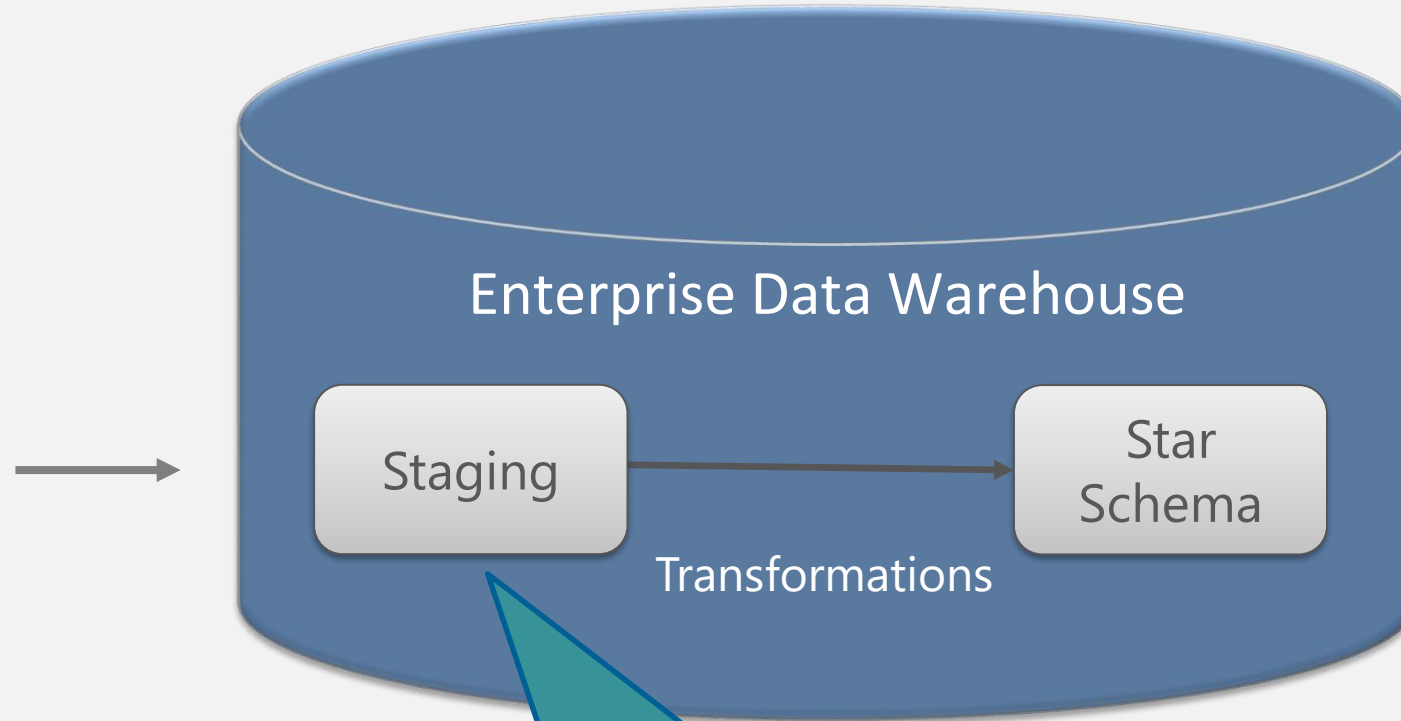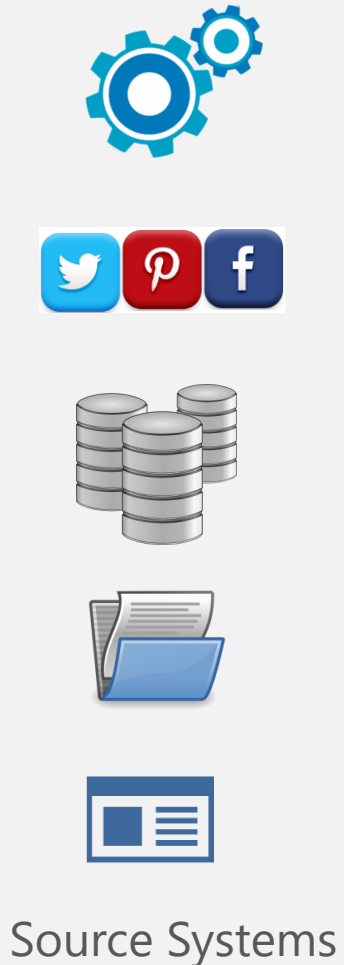**Enterprise Data Warehouse**
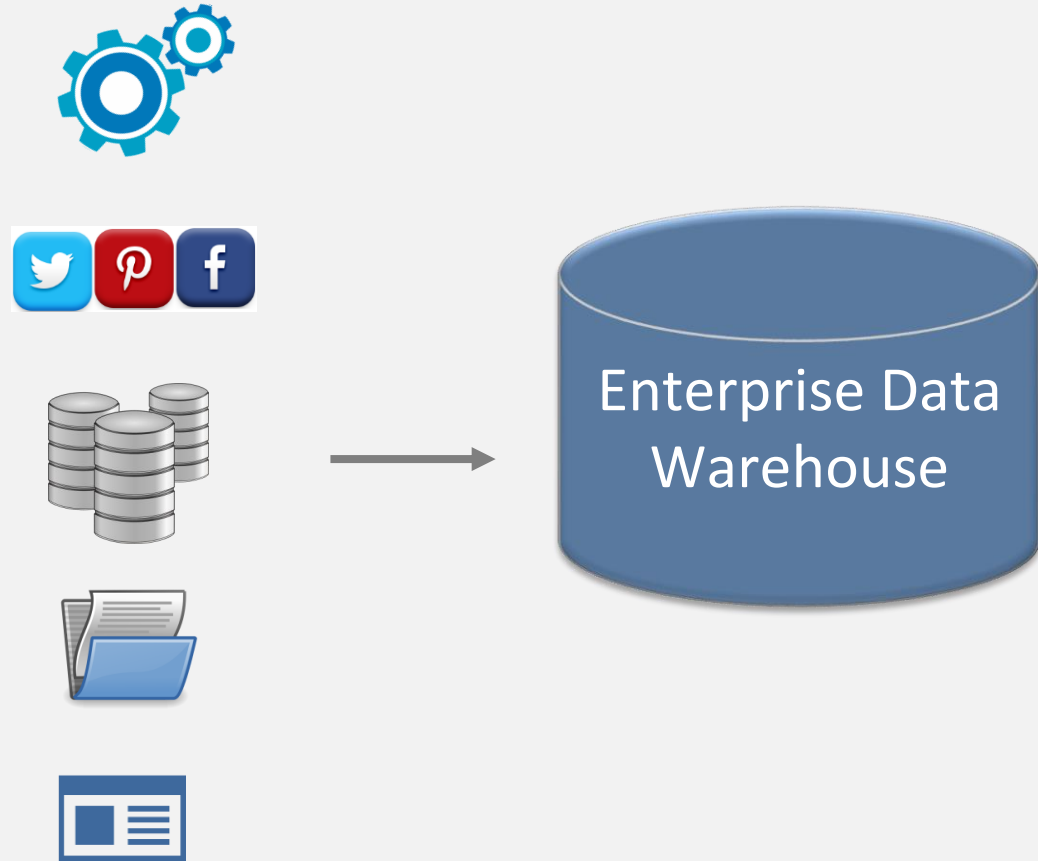
Source Systems

Objective:
Data is inherently more **valuable** once it is integrated.

Example:
Full view of a customer:
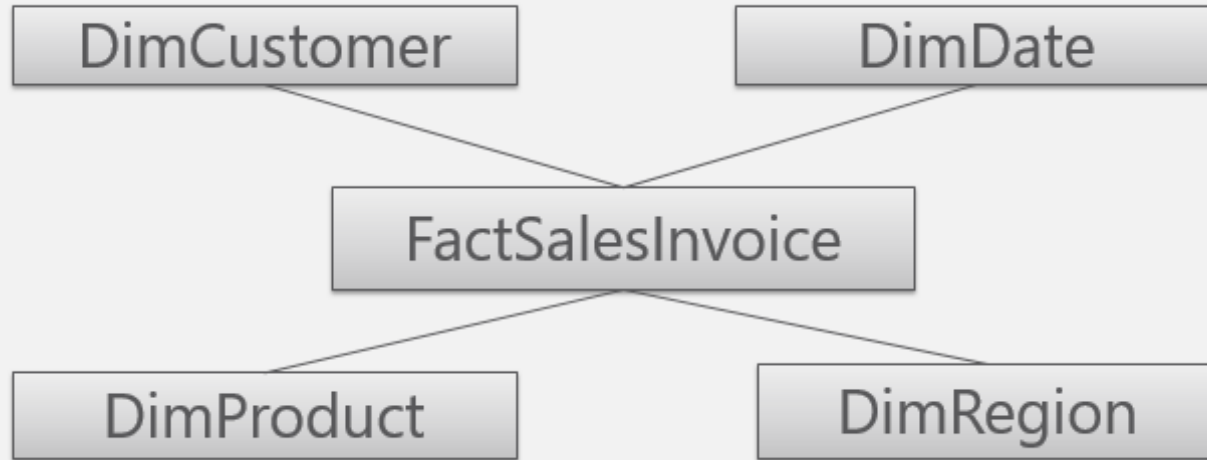- o Sales activity +
- o Delinquent invoices +
- o Support/help requests

Use of Staging Environment

Enterprise Data Warehouse

Staging → Star Schema

Transformations

Source Systems

Staging Objectives:
- ✓ **Reduce load** on source system
- ✓ No changes to source format
- ✓ A "kitchen area"
- ✓ Snapshot of source data for troubleshooting

New trend: use of a data lake as the DW staging environment

Usage of a Star Schema

```
  DimCustomer            DimDate

         FactSalesInvoice

  DimProduct             DimRegion
```

Dimension Table

Provides the **descriptive context** – attributes with the who, what, when, why, or how

Fact Table

Fact tables contain the **numeric**, **quantitative** data (aka **measures**)
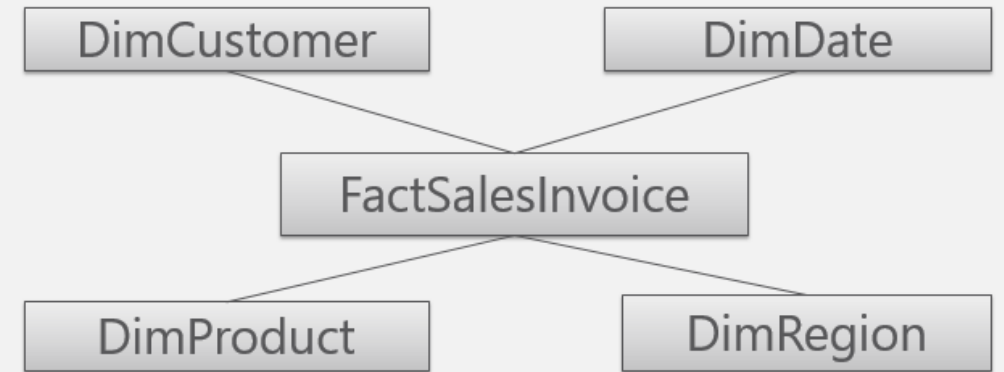
Benefits of a Star Schema

Optimal for **known** reporting scenarios

Denormalized structure, structured around **business logic**, is good for **performance** & **consistency**

Decoupled from source systems: **surrogate keys** which have no intrinsic meaning

Usability:
- ✓ Stable, predictable environment
- ✓ Less joins, easier navigation
- ✓ Friendly, recognizable names
- ✓ History retention
- ✓ Integrate multiple systems

DimCustomer

DimDate

FactSalesInvoice

DimProduct

DimRegion

Challenges of a Star Schema

Requires **up-front analysis** ("schema on write")

Difficult to handle new & **unpredictable or exploratory scenarios**

Increasing **volumes of data**

Reducing windows of time for **data loads** (near real-time is challenging)

Data quality issues are often surfaced in the reporting layer

Not practical to contain ***all* of the data** all the time

DimCustomer   DimDate

FactSalesInvoice

DimProduct   DimRegion
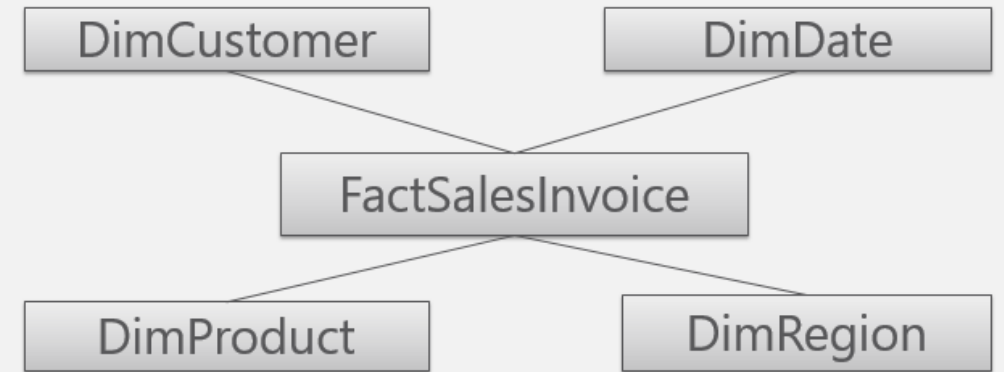
Declare Grain of Each Table
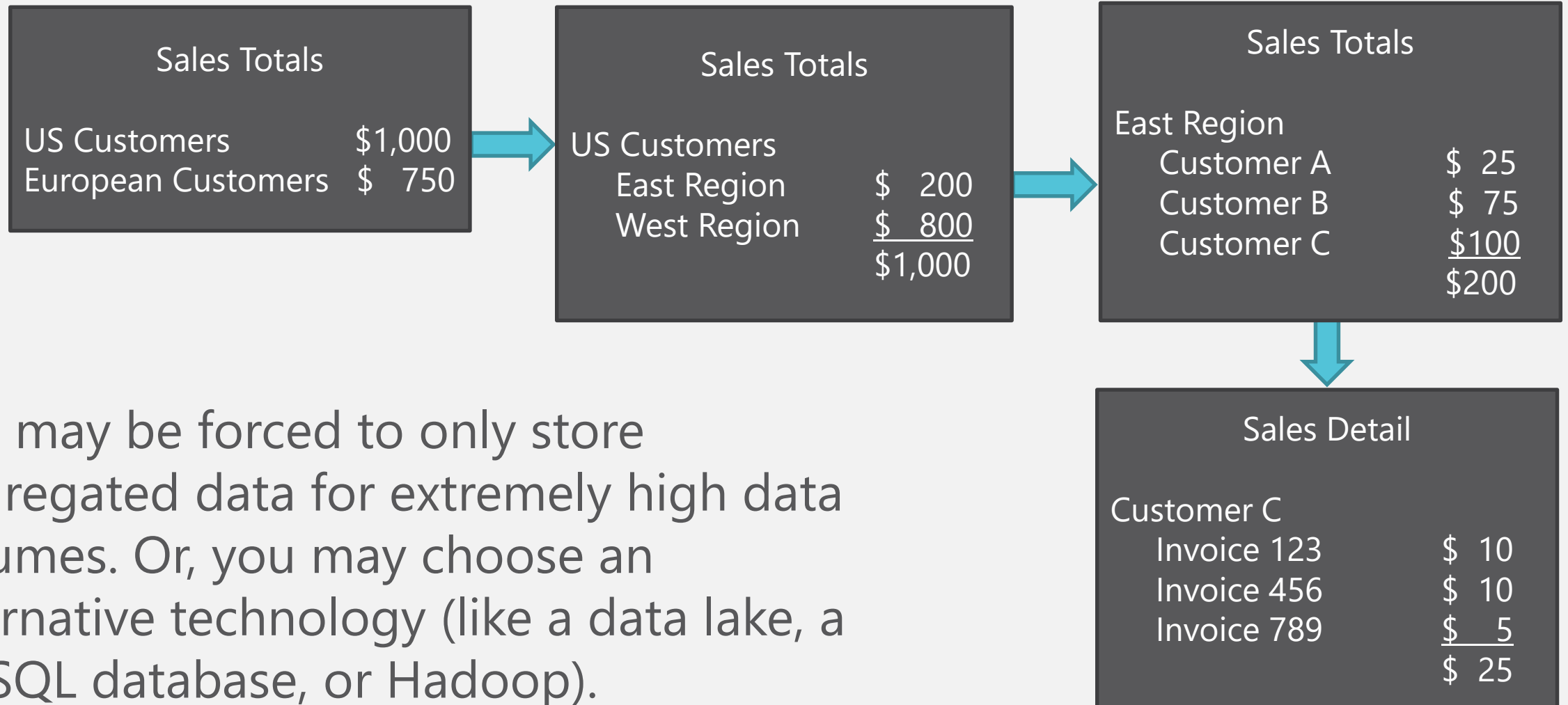
EnterpriseDW
Bus Matrix

Contributors: Melissa Coates

Last updated: 12/1/2016

Dim Table Name:	Dim Customer	Dim Region	Dim Warehouse	Dim Employee	Dim Date	Dim SalesInvoice	Dim Product
Dim Table Type:	Standard	Standard	Standard	Parent/Child	Standard	Standard	Standard
SCD Type:	Type 2	Type 1	Type 1	Type 1	Type 1	Type 1	Type 2
Dim Granularity:	One row per customer	One row per region	One row per warehouse	One row per employee (recursive)	One row per day	One row per invoice, per AR transaction	One row per product

Subject Area:	Fact Table Name:	Fact Table Type:	Fact Purpose:	Fact Granularity:							
Sales	FactSalesInvoice	Transaction Fact	All sales/AR transactions (invoices + DM, CM, cash applications, and write-offs). This transaction is updated throughout the life of the obligation.	One row per invoice, per AR transaction, per customer	X	X	X	X	X (role-playing)	X	X (via bridge table)
Accounts Receivable	FactAR SnapshotDaily	Periodic Snapshot Fact	Daily summary of open AR (retained for a rolling 5 quarters)	One row per customer, per open AR invoice, per day	X				X		
All	FactCustomer Current	Accumulating Snapshot Fact	Current summary of balances (credit limit, open order amount, AR balances, etc + current salesperson); relevant across subject areas	One row per customer	X	X	X	X (role-playing)	X (role-playing)		
Human Resources	FactEmployee Snapshot	Periodic Snapshot Fact	Annual snapshot of employee info	One row per employee, per year		X		X	X (role-playing)		

Store the Lowest Level Detail You Have

Drill-down behavior:

Sales Totals

US Customers $1,000
European Customers $ 750

Sales Totals

US Customers
    East Region $ 200
    West Region $ 800
                      $1,000

Sales Totals

East Region
    Customer A $ 25
    Customer B $ 75
    Customer C $100
                      $200

Sales Detail

Customer C
    Invoice 123 $ 10
    Invoice 456 $ 10
    Invoice 789 $ 5
                      $ 25

You may be forced to only store aggregated data for extremely high data volumes. Or, you may choose an alternative technology (like a data lake, a NoSQL database, or Hadoop).

Dimension Design

Dimension Tables

Dimension tables: provide the **descriptive context** – attributes with the who, what, when, why, or how. They should always include **friendly names & descriptions**.

Dimension tables can contain:

Type of Column in a Dim	Example
Attributes	Customer Name
Non-additive numeric value	Customer Value to Acquisition Cost Ratio
Numeric value used *only* for filtering or grouping (usually accompanied by a "band of ranges")	Customer Satisfaction % Customer Satisfaction Range 90%-100% 80-89% Less than 80%

*Dimension tables should *not* contain aggregatable numeric values (measures).*

Types of Dimension Tables

Most common types of dimensions:

Type of Dim Table	Description
Type 0	Values cannot change (ex: DimDate).
Type 1	Any value which changes is overwritten; no history is preserved.
Type 2 aka Slowly Changing Dimension	Certain important values which change generate a new row which is effective-dated. *(Not all columns should be type 2 - certain columns can be type 1.)*
Type 6	Hybrid of type 1 and 2 which includes a new column for the important values, as well as a new row.

Types 3, 4, 5, and 7 do exist, but are less commonly utilized.

Type 1 Dimension

Original data:

Customer SK	Customer NK	Customer Name	AuditRow UpdateDate
1	ABC	Brian Jones	6-4-2014
2	DEF	Sally Baker	10-1-2015

Change to Customer Name occurs.

Updated data:

Customer SK	Customer NK	Customer Name	AuditRow UpdateDate
1	ABC	Brian Jones	6-4-2014
2	DEF	**Sally Walsh**	**12-2-2016**

Type 2 Dimension

Original data:

Customer SK	Customer NK	Customer Name	AuditRow Effective Date	AuditRow Expired Date	AuditRow IsCurrent
1	ABC	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	10-1-2015	12-31-9999	1

Change to Customer Name occurs.

Updated data:

Customer SK	Customer NK	Customer Name	AuditRow Effective Date	AuditRow Expired Date	AuditRow IsCurrent
1	ABC	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	10-1-2015	**12-2-2016**	**0**
3	**DEF**	**Sally Walsh**	**12-3-2016**	**12-31-9999**	**1**

Type 6 Dimension

Original data:

Customer SK	Customer NK	Customer Name	Customer Name Current	AuditRow Effective Date	AuditRow Expired Date	Audit RowIs Current
1	ABC	Brian Jones	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	Sally Baker	10-1-2015	12-31-9999	1

Change to Customer Name occurs.

Updated data:

Customer SK	Customer NK	Customer Name	Customer Name Current	Audit Row Effective Date	AuditRow Expired Date	Audit RowIs Current
1	ABC	Brian Jones	Brian Jones	6-4-2014	12-31-9999	1
2	DEF	Sally Baker	**Sally Walsh**	**10-1-2015**	**12-2-2016**	**0**
3	**DEF**	**Sally Walsh**	**Sally Walsh**	**12-3-2016**	**12-31-9999**	**1**

Conformed Dimension

A conformed dimension **reuses the same dimension** across numerous fact tables: critical for unifying data from various sources.

Conformed dimensions provide significant value with '**drill across**' functionality, and provide a **consistent** user experience.
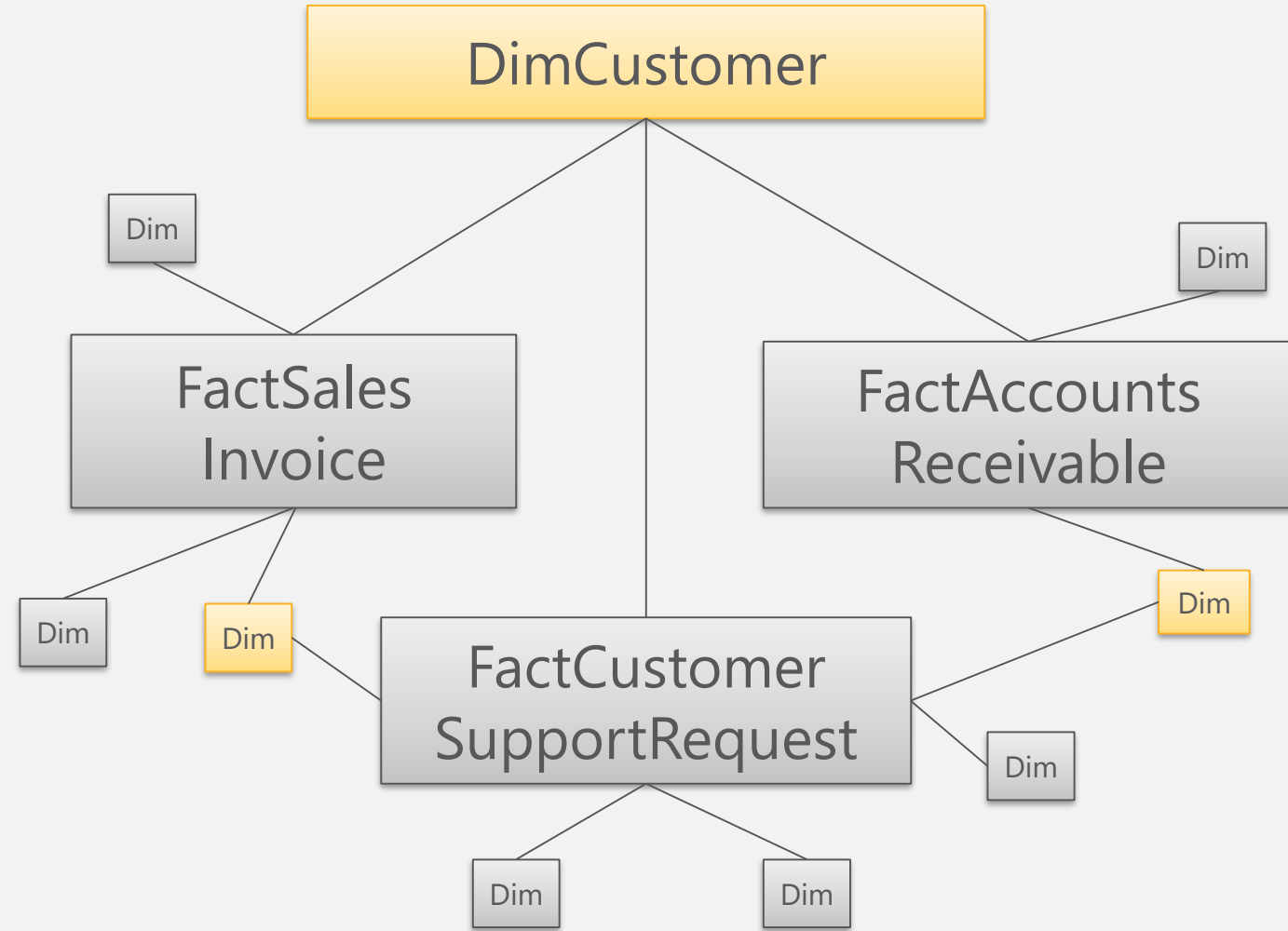
Role-Playing Dimension

A role-playing dimension utilizes the same conformed dimension. Objective is to avoid creating multiple physical copies of the same dimension table.

```
SELECT
    FSI.SalesAmount
    ,InvoiceDate = DtInv.Date
    ,PymtDueDate = DtDue.Date
FROM FactSalesInvoice AS FSI
INNER JOIN DimDate AS DtInv
    ON FSI.DateSK_InvoiceDate = DtInv.DateSK
INNER JOIN DimDate AS DtDue
    ON FSI.DateSK_PaymentDueDate = DtDue.DateSK
```

FactSalesInvoice

DateSK_InvoiceDate
DateSK_PaymentDueDate
SalesAmount

...

DimDate

DateSK
Date
Month
Quarter
Year

...
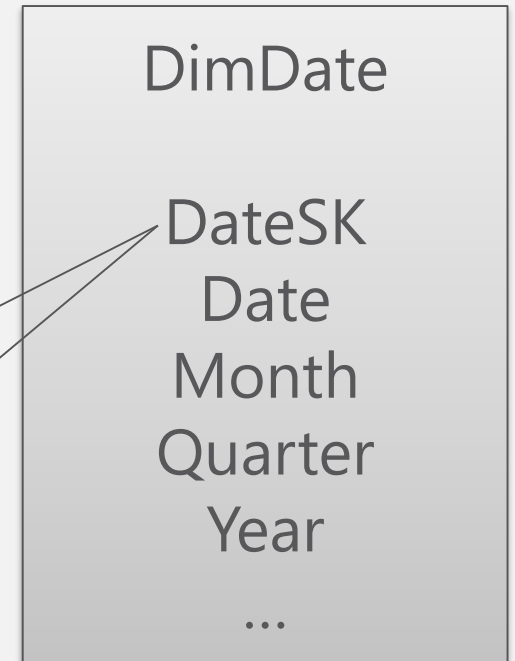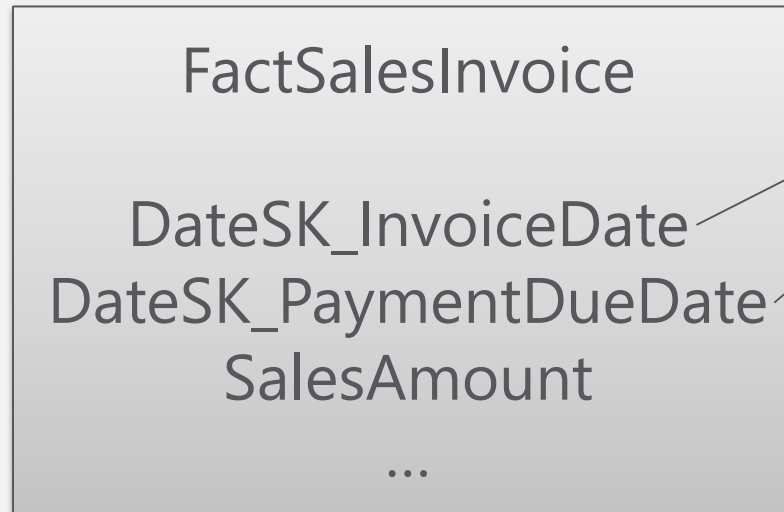
Hierarchies

Hierarchies are extremely useful for handling rollups, and for drill-down & drill-through behavior.

Date Hierarchy

Year
   Quarter
      Month
         Day

Geography Hierarchy

Country
   State or Province
      City
         Address

Dimension Design

```sql
1   CREATE TABLE [DW].[DimCustomer] (
2    [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3   ,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_RegionNumberNK] DEFAULT (N'') NOT NULL
4   ,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumberNK] DEFAULT (N'') NOT NULL
5   ,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber] DEFAULT (N'') NOT NULL
6   ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
7   ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
8   ,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
9   ,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
10  ,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
11  ,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
12  ,[CustomerTypeDe                                                                    NULL
13  ,[CustomerTy                                                               N'') NOT NULL
14  ,[CustomerTyp                                                              N'') NOT NULL
```

Inline syntax format works in the SSDT database project which requires "declarative development."

No alters beneath the create.

Dimension Design

Remove the Dim or Fact prefix from user access layers.

```sql
1   CREATE TABLE [DW].[DimCustomer] (
2    [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3   ,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_RegionNumberNK] DEFAULT (N'') NOT NULL
4   ,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumberNK] DEFAULT (N'') NOT NULL
5   ,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber] DEFAULT (N'') NOT NULL
6   ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
7   ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
8   ,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
9   ,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
10  ,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
11  ,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
12  ,[CustomerTypeDesc] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerTypeDesc] DEFAULT (N'') NOT NULL
13  ,[CustomerTypeCodeDesc] NVARCHAR(36) CONSTRAINT [dfDimCustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
14  ,[CustomerTypeDescCode] ... RAINT [dfDimCustomer_CustomerTypeDescCode] DEFAULT (N'') NOT NULL
```

Golden rule: a column exists in one and only one place in the DW.

Dimension Design

```sql
1   CREATE TABLE [DW].[DimCustomer] (
2    [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3   ,[RegionNumberNK] NVARCHAR(
4   ,[CustomerNumberNK] NVARCHAR(10) CONSTRAIN        ULL
5   ,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_
6   ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_Cus
7   ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
8   ,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
9   ,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
10  ,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
11  ,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
12  ,[CustomerTypeDesc] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerTypeDesc] DEFAULT (N'') NOT NULL
13  ,[CustomerTypeCodeDesc] NVARCHAR(36) CONSTRAINT [dfDimCustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
14  ,[CustomerTypeDescCod      RCHAR(36) CONSTRAINT [dfDimCustomer_CustomerType
```

Use a naming convention to easily identify surrogate keys & natural keys

Make careful decisions on the use of varchar vs. nvarchar

Use the smallest datatypes you can use without risk of overflows

Dimension Design

```
1    CREATE TABLE [DW].[DimCustomer] (
2     [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3    ,[RegionNumberNK] NVARCHAR(10) CONSTRAINT
4    ,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumb
5    ,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber
6    ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DE
7    ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerN
8    ,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNu
9    ,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNa
10   ,[CustomerCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocat
11   ,[CustomerTypeCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerType
12   ,[CustomerTypeDesc] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerTypeDe
13   ,[CustomerTypeCodeDesc] NVARCHAR(36) CONSTRAINT [dfDimCustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
14    CustomerTypeDescCode] DEFAULT (N'') NOT NULL
```

Alternatively, could be converted in a view or semantic layer. Objective is to avoid reporting tools trying to sum.

Avoid numeric data types for non-aggregatable columns such as Customer Number.
Also useful for retaining leading 0s or for international zip codes.

Dimension Design


```sql
1   CREATE TABLE [DW].[DimCustomer] (
2    [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3   ,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_RegionNumberNK] DEFAULT (N'') NOT NULL
4   ,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumberNK] DEFAULT (N'') NOT NULL
5   ,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCustomer_CustomerNumber] DEFAULT (N'') NOT NULL
6   ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
7   ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
8   ,[CustomerNumberName] NVARCHAR(4) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT NOT NULL
9   ,[CustomerNameNumber] NVARCHAR CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT NOT NULL
        imCustomer_AccountLocationFIPSCode] DEFAULT ) NOT NULL
        Customer_CustomerTypeCode] DEFAULT ( ULL
        Customer_Cu
        FDimCustom                                    ULL
        FDimCustom                                    ULL
```

Default constraints are present for non-nullable columns.
In a DW, defaults are optional if ETL strictly controls all data management. *Don't let SQL Server auto-name constraints.

Avoid 'Or Is Null' issues for attributes which are commonly used in predicates.

Dimension Design

```
1   CREATE TABLE [DW].[DimCustomer] (
2     [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3    ,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDi...
4    ,[CustomerNumberNK] NVARCHAR(10) CONS...
5    ,[CustomerNumber] NVARCHAR(...
6    ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustome...
7    ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
8    ,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
9    ,[CustomerNameNumber] NVAR...(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
10   ,[CustomerFIPSCode] NVARCHAR(...STRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
11   ,[CustomerTypeCode] NVARCHAR(10)...[dfDimCustomer_CustomerTypeCode] DEFAULT (N'') NOT NULL
12   ,[CustomerTypeDesc] NVARCHAR(30) C...mCustomer_CustomerTypeDesc] DEFAULT (N'') NOT NULL
13   ,[CustomerTypeCodeDesc] NVARCHAR(36) ...ustomer_CustomerTypeCodeDesc] DEFAULT (N'') NOT NULL
14   ,[CustomerTypeDescCode] NVA...                                    ...OT NULL
```

When designing a Type 2 (or 6) dimension, only choose the most important columns to generate a new row when it changes.

A 'Current' column (which is the same across all rows in a Type 6 dimension) is helpful for columns commonly used in reporting so all history shows the newest value.

Dimension Design

```sql
1   CREATE TABLE [DW].[DimCustomer] (
2    [CustomerSK] INT IDENTITY (1, 1) NOT NULL
3   ,[RegionNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCus
4   ,[CustomerNumberNK] NVARCHAR(10) CONSTRAINT [dfDimCustomer_            _NK] DEFAULT (N'') NOT NULL
5   ,[CustomerNumber] NVARCHAR(10) CONSTRAINT [dfDimCus    _CustomerNumber] DEFAULT (N'') NOT NULL
6   ,[CustomerName] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerName] DEFAULT (N'') NOT NULL
7   ,[CustomerNameCurrent] NVARCHAR(30) CONSTRAINT [dfDimCustomer_CustomerNameCurrent] DEFAULT (N'') NOT NULL
8   ,[CustomerNumberName] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNumberName] DEFAULT (N'') NOT NULL
9   ,[CustomerNameNumber] NVARCHAR(45) CONSTRAINT [dfDimCustomer_CustomerNameNumber] DEFAULT (N'') NOT NULL
10  ,[CustomerFIPSCode] NVARCHAR(10) CONSTRAINT [dfDimCustomer_AccountLocationFIPSCode] DEFAULT (N'') NOT NULL
11  ,[CustomerTypeCode] NVARCHAR(  0) CONST
12  ,[CustomerTypeDesc] NVARCHAR    CON
13  ,[CustomerTypeCodeDesc] NVARCH
14  ,[CustomerTypeDescCode] NVARCHAR
```

> Could also be derived in views or semantic layer. Or, computed columns could be used.

> Optionally, can store variations of concatenated columns such as:
> Name (Number)
> Number - Name
> Description (Code)
> Code - Description

Dimension Design

```
45   ,[CustomerDeliveryOnSaturday] NVARCHAR(
46   ,[AuditETLBatchID] INT NOT NULL
47   ,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer_AuditInsertDa
48   ,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [dfDimCustomer_A
49   ,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_AuditHashVa
50   ,[AuditModifiedDate] DATETIME NULL
51   ,[AuditModifiedBy] NVARCHAR(50) NULL
52   ,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDeleted] DE
53   ,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_AuditIsPurgeE
54   ,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCustomer_RowEffec
55   ,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDimCustomer_RowExpiredDate] DEFAULT ('12/31/2999') NOT NULL
56   ,[AuditRowIsCurrent]    BIT CONSTRAINT [dfDimDimCustomer_RowIsCurrent] DEFAULT ((0)) NOT NULL
57   ,CONSTRAINT [pkCustomerSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
58   ,CONSTRAINT [uqDimCus    ] UNIQUE NONCLUSTERED ([CustomerNumberNK] ASC,
59                                                  [RegionNumberNK] ASC,
60                                                  [AuditRowEffectiveDate] ASC,
61                                                              ON [Dimensions]
62   ) ON [Dimensions
```

Standard audit columns.
The 'Audit' prefix makes it clear they are generated in the DW not the source.

Additional columns if the Type 2 historical change tracking is occurring.

Dimension Design

All key & index suggestions are merely a starting point. As your DW grows, you might have to refine your strategy depending on ETL.

```
44  ...merDe...                ...VARCHA...
45  ,[CustomerDeliveryOnSaturday] NVARCHAR(3) CONSTRAINT [df...
46  ,[AuditETLBatchID] INT NOT NULL CONSTRAINT [dfDimCustome...
47  ,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer_Au...
48  ,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [dfDim...
49  ,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_A...
50  ,[AuditModifiedDate] DATETIME NULL
51  ,[AuditModifiedBy] NVARCHAR(50) NULL
52  ,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDeleted] DEFAULT ((0)) NOT NULL
53  ,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_AuditIsPurgeEligible] DEFAULT ((0)) NOT NULL
54  ,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCustomer_RowEffectiveDate] DEFAULT (SYSDATETIME()) NOT NULL
55  ,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDimCustomer_RowExpiredDate] DEFAULT ('12/31/2999') NOT NULL
56  ,[AuditRowIsCurrent]   BIT CONSTRAINT [dfDimDimCustomer_RowIsCurrent] DEFAULT ((0)) NOT NULL
57  ,CONSTRAINT [pkCustomerSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
58  ,CONSTRAINT [uqDimCustomer] UNIQUE NONCLUSTERED ([CustomerNumberNK] ASC,
59                                                  [RegionNumberNK] ASC,
60                                                  [AuditRowEffectiveDate] ASC,
61                                                  [AuditRowExpiredDate] ASC) ON [Dimensions]
62  ) ON [Dimensions];
```

Primary key based on the surrogate key. This is also our clustered index.

Dimension Design

```
45  ,[CustomerDeliveryOnSaturday] NVARCHAR(3) CONSTRAINT [dfD
46  ,[AuditETLBatchID] INT NOT NULL CONSTRAINT [dfDimCustomer
47  ,[AuditInsertDate] DATETIME CONSTRAINT [dfDimCustomer_Aud
48  ,[AuditInsertBy] NVARCHAR(50) NOT NULL CONSTRAINT [dfDimC
49  ,[AuditHashValue] BINARY(20) CONSTRAINT [dfDimCustomer_Aud
50  ,[AuditModifiedDate] DATETIME NULL
51  ,[AuditModifiedBy] NVARCHAR(50) NULL
52  ,[AuditIsDeleted] BIT CONSTRAINT [dfDimCustomer_AuditIsDelet        ((0)) NOT NULL
53  ,[AuditIsPurgeEligible] BIT CONSTRAINT [dfDimCustomer_Aud        ligible] DEFAULT ((0)) NOT NULL
54  ,[AuditRowEffectiveDate] DATETIME CONSTRAINT [dfDimCus      RowEffectiveDate]  DEFAULT (SYSDATETIME()) NOT NULL
55  ,[AuditRowExpiredDate] DATETIME CONSTRAINT [dfDimDi    tomer_RowExpiredDate]  DEFAULT ('12/31/2999') NOT NULL
56  ,[AuditRowIsCurrent]    BIT CONSTRAINT [dfDimDi   stomer_RowIsCurrent]  DEFAULT ((0)) NOT NULL
57  ,CONSTRAINT [pkCustomerSK] PRIMARY KEY CLUSTERED ([CustomerSK] ASC) ON [Dimensions]
58  ,CONSTRAINT [uqDimCustomer] UNIQUE NONCLUSTERED ([CustomerNumberNK] ASC,
59                                                   [RegionNumberNK] ASC,
60                                                   [AuditRowEffectiveDate] ASC,
61                                                   [AuditRowExpiredDate] ASC) ON [Dimensions]
62  ) ON [Dimensions];
```

The unique constraint implicitly creates a unique index as well, which will assist with lookup operations.

Unique constraint, based on natural keys, defines the "grain" of the table. It also helps identify data quality issues & is very helpful to the SQL Server query optimizer.

Dimension Design

Fact Design

Fact Tables

Fact tables contain the **numeric**, **quantitative** data (aka **measures**).

Typically **one fact table per distinct business process.**
Exception: "consolidated" facts (aka "merged" facts) such as actual vs. forecast which require the same granularity and are frequently analyzed together.

Fact tables can contain:

Type of Column in a Fact	Example
Measures	Sales Amount
Foreign keys to dimension table	3392 (meaningless integer surrogate key)
Degenerate dimension	Order Number

Types of Fact Tables

Most common types of facts:

Type of Fact Table	Description	Example
Transaction Fact	An event at a point in time	FactSalesInvoice
Periodic Snapshot Fact	Summary at a point in time	FactARBalanceDaily
Accumulating Snapshot Fact	Summary across the lifetime of an event	FactStudentApplication
Timespan Tracking Fact	Effective-dated rows	FactCapitalAssetBalance

Other facts:

Type of Fact Table	Description	Example
Factless Fact Table	Recording when an event did not occur	FactPromotionNoSales
Aggregate Facts	Rollups, usually to improve reporting speed	FactSalesInvoiceSummary

Fact Design

```sql
CREATE TABLE DW.FactSalesInvoice(
SalesInvoiceSK INT
    CONSTRAINT dfFactSalesInvoice_ARObligation
    CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
,CustomerSK INT
    CONSTRAINT dfFactSalesInvoice_...SK DEFAULT ((-1)) NOT NULL
    CONSTRAINT fkFactSalesInvoice_Di... REFERENCES DW.DimCustomer(CustomerSK)
,RegionSK SMALLINT
    CONSTRAINT dfFactSalesInvoi...
    CONSTRAINT fkFactSalesInvoi...
,DateSK_AROpenedDate INT
    CONSTRAINT dfFactSalesInvoi...
    CONSTRAINT fkFactSalesInvoi...                    SK)
,DateSK_ARClosedDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
    CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
,DateSK_ARDiscountDate INT
    CONSTRAINT dfFactSalesInvoice_DateSK_ARDiscountDate DEFAULT(29991231) NOT NULL
```

> Even if all of the SKs are the same, avoid combining fact tables for unrelated business processes.

> One fact table per distinct business process.

Fact Design

> The combination of SKs might dictate the grain of the fact table, but it may not.

```sql
1   CREATE TABLE DW.FactSalesInvoice
2   SalesInvoiceSK INT
3       CONSTRAINT dfFactSalesInvoice_ARObligationSK DEFAULT ((-1)) NOT NULL
4       CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
5   ,CustomerSK INT
6       CONSTRAINT dfFactSalesInvoice_CustomerSK DEFAULT ((-1)) NOT NULL
7       CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK)
8   ,RegionSK SMALLINT
9       CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((-1)) NOT NULL
10      CONSTRAINT fkFactSalesInvoice_DimRegion REFERENCES DW.DimRegion(RegionSK)
11  ,DateSK_AROpenedDate INT
12      CONSTRAINT dfFactSalesInvoice_DateSK_AROpenedDate DEFAULT(19000101) NOT NULL
13      CONSTRAINT fkFactSalesInvoice_DimDate_AROpenedDate REFERENCES DW.DimDate(DateSK)
14  ,DateSK_ARClosedDate INT
15      CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
16      CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
17  ,DateSK_ARDiscountDate INT
18      CONSTRAINT dfFactSalesInvoice_DateSK_ARDiscountDate DEFAULT(29991231) NOT NULL
19
```

Fact Design

```sql
1   CREATE TABLE DW.FactSalesInvoice
2   SalesInvoiceSK INT
3       CONSTRAINT dfFactSalesInvoice_ARObligationSK DEFAULT ((-1)) NOT NULL
4       CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
5   ,CustomerSK INT
6       CONSTRAINT dfFactSalesInvoice_CustomerSK DEFAULT ((-1)) NOT NULL
7       CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK)
8   ,RegionSK SMALLINT
9       CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((-1)) NOT NULL
10      CONSTRAINT fkFactSalesInvoice_DimRegion REFERENCES DW.DimRegion(RegionSK)
11  ,DateSK_AROpenedDate INT
12      CONSTRAINT dfFactSalesInvoice_DateSK_AROpenedDate DEFAULT(19000101) NOT NULL
13      CONSTRAINT fkFactSalesInvoice_DimDate_AROpenedDate REFERENCES DW.DimDate(DateSK)
14  ,DateSK_ARClosedDate INT
15      CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
16      CONSTRAINT fkFactSales_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
17  ,DateSK_ARDiscountDate
18      CON
19      CON
```

Some data modelers prefer the unknown member row to have its key assigned randomly.

Default equates to the 'unknown member' row.

Fact Design

```sql
1   CREATE TABLE DW.FactSalesInvoice(
2   SalesInvoiceSK INT
3       CONSTRAINT dfFactSalesInvoice_AR
4       CONSTRAINT fkFactSalesInvoice_Di               eSK)
5   ,CustomerSK INT
6       CONSTRAINT dfFactSalesInvoice_Cu
7       CONSTRAINT fkFactSalesInvoice_DimC
8   ,RegionSK SMALLINT
9       CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT (        NULL
10      CONSTRAINT fkFactSalesInvoice_DimRegion REFERENCES D    Region(RegionSK)
11  ,DateSK_AROpenedDate INT
12      CONSTRAINT dfFactSalesInvoice_DateSK_AROpenedDate DEFAULT(19000101) NOT NULL
13      CONSTRAINT fkFactSalesInvoice_DimDate_AROpenedDate REFERENCES DW.DimDate(DateSK)
14  ,DateSK_ARClosedDate INT
15      CONSTRAINT dfFactSalesInvoice_DateSK_ARClosedDate DEFAULT(29991231) NOT NULL
16      CONSTRAINT fkFactSalesInvoice_DimDate_ARClosedDate REFERENCES DW.DimDate(DateSK)
17  ,Dat
18
19
```

Optionally can use two types of Date defaults: one in the past, one in the future. Helps with 'Less than' or 'Greater than' predicates.

It's also fine for a date SK to be an actual date datatype instead of an integer.

Fact Design


```sql
1   CREATE TABLE DW.FactSalesInvoice(
2   SalesInvoiceSK INT
3       CONSTRAINT dfFactSalesInvoice_ARObligationSK DEFAULT ((-1)) NOT NULL
4       CONSTRAINT fkFactSalesInvoice_DimSalesInvoice REFERENCES DW.DimSalesInvoice(SalesInvoiceSK)
5   ,CustomerSK INT
6       CONSTRAINT dfFactSalesInvoice_CustomerSK DEFAULT ((-1)) NOT NULL
7       CONSTRAINT fkFactSalesInvoice_DimCustomer REFERENCES DW.DimCustomer(CustomerSK)
8   ,RegionSK SMALLINT
9       CONSTRAINT dfFactSalesInvoice_RegionSK DEFAULT ((  )) NOT NULL
10      CONSTRAINT fkFactSalesInvoice_DimRegion REFEREN    DW.DimRegion(RegionSK)
11  ,DateSK_AROpenedDate INT
12      CONSTRAINT dfFactSalesInvoice_DateSK_AROpen        EFAULT(19000101) NOT NULL
13      CONSTRAINT fkFactSalesInvoice_DimDate_AROp         REFERENCES DW.DimDate(DateSK)
14  ,DateSK_ARClosedDate INT
15      CONSTRAINT dfFactSalesInvoice_DateSK_A             EFAULT(29991231) NOT NULL
16      CONSTRAINT fkFactSalesInvoice_DimDate              REFERENCES DW.DimDate(DateSK)
17  ,DateSK_
18      CONS                                              29991231) NOT NULL
19      CONS
```

Having a PK in a fact is personal preference. Usually you don't want a clustered index on it though.

Foreign key constraints mitigate referential integrity issues.

Fact Design

```
 23    ,WarehouseSK INT
 24        CONSTRAINT dfFactSalesInvoice_WarehouseSK DEFAULT ((-1)) NOT NULL
 25        CONSTRAINT fkFactSalesInvoice_DimWarehouse REFERENCES DW.DimWarehouse(WarehouseSK)
 26    ,ProductSK INT
 27        CONSTRAINT dfFactSalesInvoice_ProductSK DEFAULT ((-1)) NOT NULL
 28        CONSTRAINT fkFactSalesInvoice_DimProduct REFERENCES DW.DimProduct(ProductSK)
 29    ,EmployeeSK_SalespersonForTransaction INT
 30        CONSTRAINT dfFactSalesInvoice_EmployeeSK_SalespersonForTransaction DEFAULT ((-1)) NOT NULL
 31        CONSTRAINT fkFactSalesInvoice_DimEmployee_SalespersonForTransaction REFERENCES DW.DimEmployee(EmployeeSK)
 32    ,NumberOfInvoices INT NULL
 33    ,TotalInvoiceAmount DECIMAL(10,2) NULL
 34    ,GrossSalesAmount DECIMAL(10,2) NULL
 35    ,FreightAmount DECIMAL(10,2) NULL
 36    ,TaxAmount DECIMAL(10,2) NULL
 37    ,NetSalesAmount DECIMAL(10,2) NULL
 38    ,DiscountAmount DECIMAL(10,2) NULL
 39    ,InvoiceNumberNK NVARCHAR(10) CONSTRAINT dfFa...              NOT NULL
 40    ,InvoiceItemNumberNK NVARCHAR(10) CONSTRAINT                  T NULL
```

Measures are sparse, therefore nullable.
0s are not stored except in a factless fact table.

Fact Design

```
47  ,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFactSalesInvoice_WarehouseNumberNK DEFAULT (N'') NOT NULL
48  ,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSalesInvoice_RegionNumberNK DEFAULT (N'') NOT NULL
49  ,AuditETLBatchID INT NOT NULL CONSTRAINT dfFactSalesInvoice_AuditKey DEFAULT ((0))
50  ,AuditInsertDate DATETIME
51  ,AuditInsertBy NVARCHAR(5
52  ,AuditHashValue BINA
53  ,AuditModifiedDate DA
54  ,AuditModifiedBy NVARCH
55  ,AuditIsDeleted BIT CONS
56  ,AuditIsPurgeEligible BI
57  ,CONSTRAINT [uqFactSales
58      ([CustomerNumberNK] A
59  ,INDEX [ixFactSalesInvoi
60      WITH (FILLFACTOR = 80
61  ,INDEX [ixFactSalesInvoi
62  ,INDEX [ixFactSalesInvoi
63  ,INDEX [ixFactSalesInvoi
64  ,INDEX [ixFactSalesInvoice_W
65  ,INDEX [ixFactSalesInvoice_EmployeeSK_SalespersonForTransaction] NONCLUSTERED (EmployeeSK_SalespersonForTransaction ASC)
66      WITH (
67  ,INDEX [i                                                                                80) ON [Facts]
68  ,INDEX [i                                                                                ON [Facts]
69  ,INDEX [i                                                                                ON [Facts]
70  ) ON [Fact
```

Natural key in a fact violates Kimball rules.
However, they are helpful for:
(1) Re-assigning SK if a lookup issue occurred and an unknown member got assigned.
(2) Allows unique constraint on the NKs for ensuring data integrity.

Never (ever!) let the NKs be exposed or used for anything besides ETL. And only create minimum # of NKs to identify the row.

Fact Design


```
47    ,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFactSalesInvoi
48    ,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSalesInvoice_R
49    ,AuditETLBatchID INT NOT NULL CONSTRAINT dfFactSalesInvoice
50    ,AuditInsertDate DATETIME CONSTRAINT  dfFactSalesInvoice_Au
51    ,AuditInsertBy NVARCHAR(50) NOT NULL CONSTRAINT dfFactSales
52    ,AuditHashValue BINARY(20) CONSTRAINT dfFactSalesInvoice_Au
53    ,AuditModifiedDate DATETIME NULL
54    ,AuditModifiedBy NVARCHAR(50) NULL
55    ,AuditIsDeleted BIT CONSTRAINT dfFactSalesInvoice_AuditIsDel      DEFAULT ((0)) NOT NULL
56    ,AuditIsPurgeEligible BIT CONSTRAINT dfFactSalesInvoice_AuditIsPurgeEligible DEFAULT ((0)) NOT NULL
57    ,CONSTRAINT [uqFactSalesInvoice] UNIQUE NONCLUSTERED
58       ([CustomerNumberNK] ASC, [InvoiceNumberNK] ASC, [InvoiceItemNumberNK] ASC) ON [Facts]
59    ,INDEX [ixFactSalesInvoic_ DateSK_AROpenedDate] CLUSTERED (DateSK_AROpenedDate ASC)
60       WITH (FILLFACTOR = 80,     COMPRESSION = PAGE) ON [Facts]
61    ,INDEX [ixFactSalesInvoice_       iceSK] NONCLUSTERED (SalesInvoiceSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
62    ,INDEX [ixFactSalesInvoice_C      ONCLUSTERED (CustomerSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
63    ,INDEX [ixFactSalesInvoice_Re         TERED (RegionSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
64    ,INDEX [ixFactSalesInvoice_War        RED (WarehouseSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
65    ,INDEX [ixFactSale                        NONCLUSTERED (          SK S           Transaction ASC)
66       WITH (FILLF
67    ,INDEX [ixFactS                                                     R = 80) ON [Facts]
68    ,INDEX [ixFactS                                                    80) ON [Facts]
69    ,INDEX [ixFactS                                                    80) ON [Facts]
70    ) ON [Facts]
```

The unique constraint implicitly creates a unique index as well, which will assist with lookup operations.

Unique constraint, based on natural keys, defines the "grain" of the table & helps identify data quality issues.

Fact Design


```
47    ,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFa...                    NULL
48    ,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSa...
49    ,AuditETLBatchID INT NOT NULL CONSTRAINT dfFact...
50    ,AuditInsertDate DATETIME CONSTRAINT  dfFactSa...                     NOT NULL
51    ,AuditInsertBy NVARCHAR(50) NOT NULL CONSTRAINT...                    SNAME())
52    ,AuditHashValue BINARY(20) CONSTRAINT dfFactSalesInvoi...   ue DEFAULT ((0)) NOT NULL
53    ,AuditModifiedDate DATETIME NULL
54    ,AuditModifiedBy NVARCHAR(50) NULL
55    ,AuditIsDeleted BIT CONSTRAINT dfFactSalesInvoice_Au...  leted DEFAULT ((0)) NOT NULL
56    ,AuditIsPurgeEligible BIT CONSTRAINT dfFactSalesInvo...  uditIsPurgeEligible DEFAULT ((0)) NOT NULL
57    ,CONSTRAINT [uqFactSalesInvoice] UNIQUE NONCLUSTERE...
58        ([CustomerNumberNK] ASC, [InvoiceNumberNK] ASC, [InvoiceItemNumberNK] ASC) ON [Facts]
59    ,INDEX [ixFactSalesInvoice_DateSK_AROpenedDate] CLUSTERED (DateSK_AROpenedDate ASC)
60        WITH (FILLFACTOR = 80, DATA_COMPRESSION = PAGE) ON [Facts]
61    ,INDEX [ixFactSalesInvoice_SalesInvoice...] NONCLUSTERED (SalesInvoiceSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
62    ,INDEX [ixFactSalesInvoice_CustomerSK] NO...USTERED (CustomerSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
63    ,INDEX [ixFactSalesInvoice_RegionSK] NON...D (RegionSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
64    ,INDEX [ixFactSalesInvoice_WarehouseSK] N...R (WarehouseSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
65    ,INDEX [ixFactSalesInvoice_EmployeeSK_Sal...ransaction] NONCLUSTERED (EmployeeSK_SalespersonForTransaction ASC)
66        WITH (FILLFACTOR = 80) ON [Fa...
67    ,INDEX [ixFactSalesInvoice_DateS...                            C) WITH (FILLFACTOR = 80) ON [Facts]
68    ,INDEX [ixFactSalesInvoice_DateS...                            ITH (FILLFACTOR = 80) ON [Facts]
69    ,INDEX [ixFactSalesInvoice_DateS...                            ITH (FILLFACTOR = 80) ON [Facts]
70    ) ON [Facts]  ;
```

The clustered index is usually on a date.

Compression set on the clustered index rather than the table.

Fact Design

```sql
47    ,WarehouseNumberNK NVARCHAR(10) CONSTRAINT dfFactSal
48    ,RegionNumberNK NVARCHAR(3) CONSTRAINT dfFactSalesIn
49    ,AuditETLBatchID INT NOT NULL CONSTRAINT dfFactSales
50    ,AuditInsertDate DATETIME CONSTRAINT  dfFactSalesInv
51    ,AuditInsertBy NVARCHAR(50) NOT NULL CONSTRAINT dfFa
52    ,AuditHashValue BINARY(20) CONSTRAINT dfFactSalesInv
53    ,AuditModifiedDate DATETIME NULL
54    ,AuditModifiedBy NVARCHAR(50) NULL
55    ,AuditIsDeleted BIT CONSTRAINT dfFactSalesInvoice_Aud
56    ,AuditIsPurgeEligible BIT CONSTRAINT dfFactSalesInvoice_
57    ,CONSTRAINT [uqFactSalesInvoice] UNIQUE NONCLUSTERED
58        ([CustomerNumberNK] ASC, [InvoiceNumberNK] ASC, [Inv      mberNK] ASC) ON [Facts]
59    ,INDEX [ixFactSalesInvoice_DateSK_AROpenedDate] CLUS        DateSK_AROpenedDate ASC)
60        WITH (FILLFACTOR = 80, DATA_COMPRESSION = PA  E) ON [Facts]
61    ,INDEX [ixFactSalesInvoice_SalesInvoiceSK] NONCLUSTERED (SalesInvoiceSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
62    ,INDEX [ixFactSalesInvoice_CustomerSK] NONCLUSTERED (CustomerSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
63    ,INDEX [ixFactSalesInvoice_RegionSK] NONCLUSTERED (RegionSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
64    ,INDEX [ixFactSalesInvoice_WarehouseSK] NONCLUSTERED (WarehouseSK ASC) WITH (FILLFACTOR = 80) ON [Facts]
65    ,INDEX [ixFactSalesInvoice_EmployeeSK_SalespersonForTransaction] NONCLUSTERED (EmployeeSK_SalespersonForTransaction ASC)
66        WITH (FILLFACTOR = 80) ON [Facts]
67    ,INDEX [ixFactSalesInvoice_DateSK_ARDiscountDate] NONCLUSTERED (DateSK_ARDiscountDate ASC) WITH (FILLFACTOR = 80) ON [Facts]
68    ,INDEX [ixFactSalesInvoice_DateSK_ARNetDueDate] NONCLUSTERED (DateSK_ARNetDueDate ASC) WITH (FILLFACTOR = 80) ON [Facts]
69    ,INDEX [ixFactSalesInvoice_DateSK_ARClosedDate] NONCLUSTERED (DateSK_ARClosedDate ASC) WITH (FILLFACTOR = 80) ON [Facts]
70    ) ON [Facts]  ;
```

Nonclustered index on each surrogate key. Useful for smaller fact tables (which don't justify a clustered columnstore index).

When to Use
Columnstore Indexes or Partitioning

Handling Larger Fact Tables

Clustered Columnstore Index

Useful for:
- ✓ Reducing data storage due to compression of redundant values
- ✓ Improving query times for large datasets
- ✓ Improving query times due to reduced I/O (ex: column elimination)

Table Partitioning

Useful for:
- ✓ Improving data load times due to partition switching
- ✓ Flexibility for maintenance on larger tables
- ✓ Improving query performance *(possibly)* due parallelism & partition elimination behavior

Clustered Columnstore Index

Simplified & conceptual

Rowstore:

CounterName	Disk	DateMeasurementTaken	TimeMeasurementTaken	Measurement
Avg. Disk sec/Read	G:\	1/30/2017	4:48:41 PM	0.01818
Avg. Disk sec/Read	L:\	1/30/2017	4:48:41 PM	0.00385
Avg. Disk sec/Read	T:\	1/30/2017	4:48:41 PM	0.00780
Avg. Disk Bytes/Read	G:\	1/30/2017	4:48:41 PM	53120.73782
Avg. Disk Bytes/Read	L:\	1/30/2017	4:48:41 PM	42362.51095
Avg. Disk Bytes/Read	T:\	1/30/2017	4:48:41 PM	47951.40657

Columnstore:

Page

CounterName
Avg. Disk sec/Read
Avg. Disk Bytes/Read

Page

Disk
G:\
L:\
T:\

Page

DateMeasurementTaken
1/30/2017

Page

TimeMeasurementTaken
4:48:41 PM

Page

Measurement
0.01818
0.00385
0.00780
53120.73782
42362.51095
47951.40657

Reduced storage for low cardinality columns

Clustered Columnstore Index

Simplified & conceptual

Page		Page		Page		Page		Page
CounterName		**Disk**		**DateMeasurementTaken**		**TimeMeasurementTaken**		**Measurement**
Avg. Disk sec/Read		G:\		1/30/2017		4:48:41 PM		0.01818
Avg. Disk Bytes/Read		L:\						0.00385
		T:\						0.00780
								53120.73782
								42362.51095
								47951.40657

CCI most suitable for:

✓ Tables over 1 million rows

✓ Data structured in a denormalized star schema format (DW not OLTP)
✓ Support for analytical query workload which scans a large number of rows, and retrieves few columns
✓ Data which is not frequently updated ('cold' data not 'hot')
✓ Can selectively be used on insert-oriented workloads (ex: IoT)

(A <u>non</u>clustered columnstore index targets analytical queries on an OLTP rather than a data warehouse.)

Partitioned Table

Useful for:
- ✓ Speeding up ETL processes
  - ✓ Large datasets (50GB+)
  - ✓ Small maintenance windows
  - ✓ Use of a sliding window
- ✓ Storage of partitions on separate drives (filegroups)
  - ✓ Older (cold) data on cheaper storage
  - ✓ Historical data on read-only filegroup
- ✓ Speeding up queries *(possibly)*
  - ✓ Partition elimination
  - ✓ Parallelism

Table A

Partition 1	Partition 2	Partition 3
Current Data	Current-1 Data	Current-2 Data

Filegroup 1	Filegroup 2	Filegroup 3

High-end storage	Slower storage
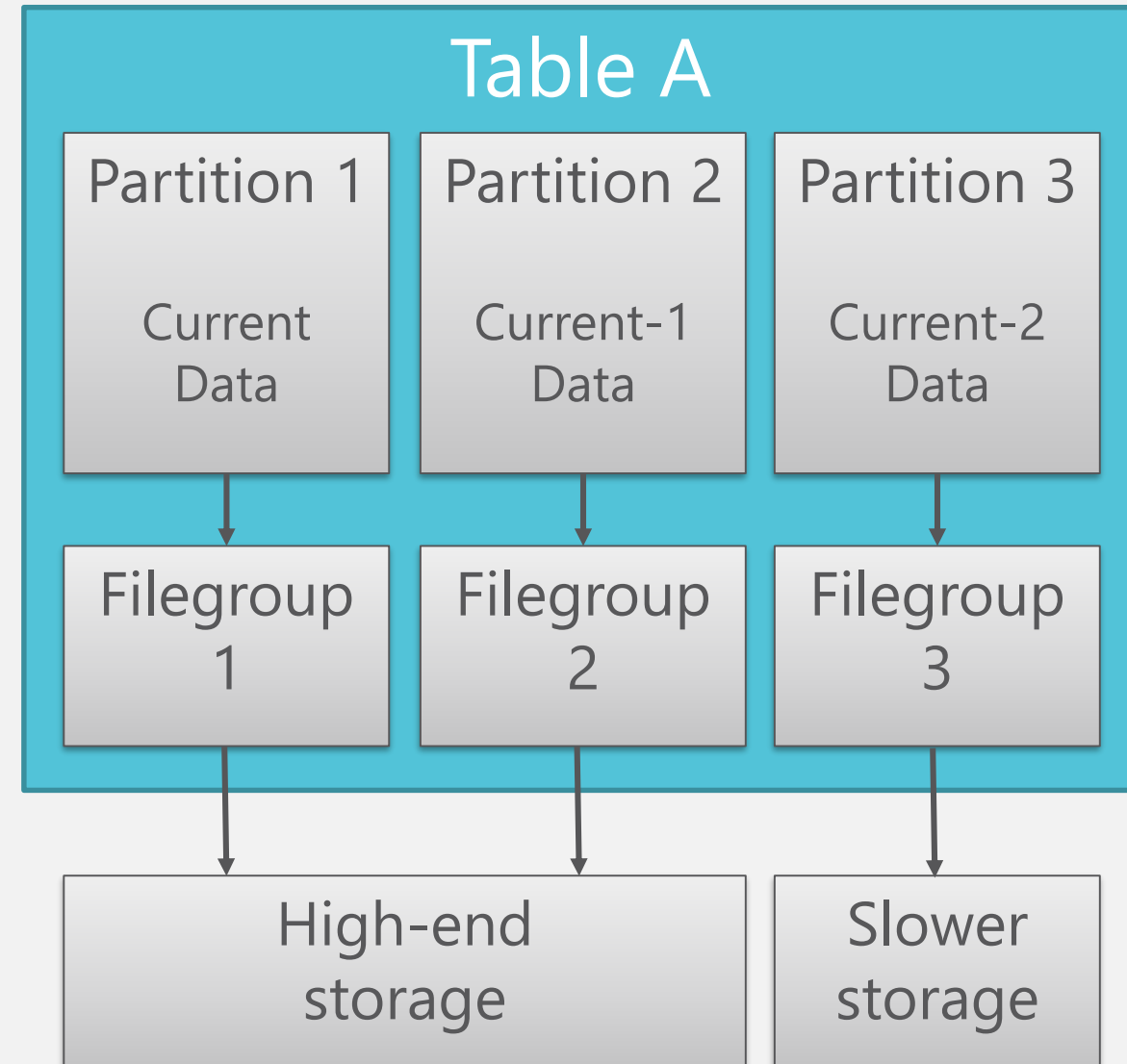
Partitioned View

Useful for:
- ✓ Query performance (similar to partitioned table)
- ✓ Sharing of a single table ("partition") across multiple views
- ✓ Displaying info from > 1 database or server (via a linked server)

> Requires "Check" constraints on the underlying tables (usually on a date column)

```sql
1  CREATE VIEW DW.vwFactSales
2  WITH SCHEMABINDING
3  AS
4
5  SELECT SalesInvoiceSK
6        ,CustomerSK
7        ,DateSK_AROpenedDate
8        ,TotalInvoiceAmount
9        ,GrossSalesAmount
10 FROM DW.FactSalesCurrent
11
12 UNION ALL
13
14 SELECT SalesInvoiceSK
15        ,CustomerSK
16        ,DateSK_AROpenedDate
17        ,TotalInvoiceAmount
18        ,GrossSalesAmount
19 FROM DW.FactSalesHistory;
20
```

Data Warehouse Tips

Handling Many-to-Many Scenarios

Classic many-to-many scenarios:

- ✓ A sales order is for many products, and a product is on many sales orders
- ✓ A customer has multiple bank accounts, and a bank account belongs to multiple customers
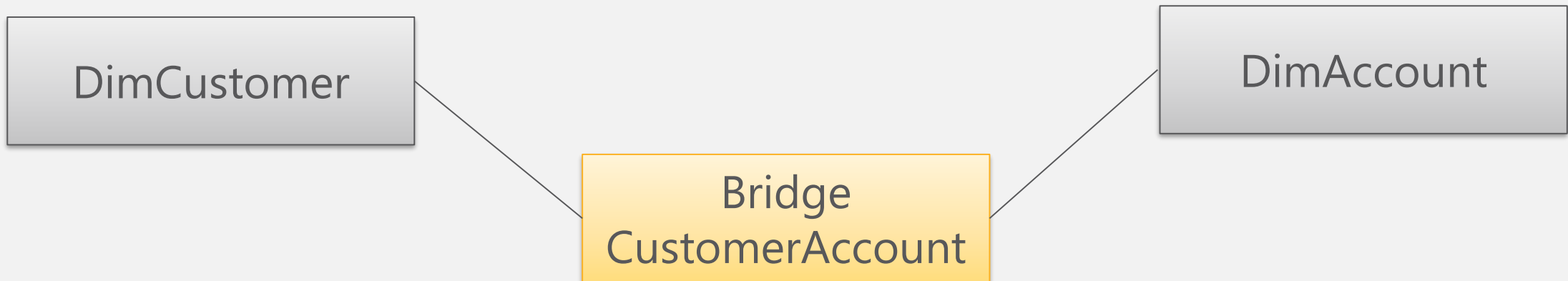
Ways to Track History in a DW

Most common options for tracking history:
1. Slowly changing dimension
2. Fact snapshot tables
3. Timestamp tracking fact

New option in SQL Server 2016:
4. Temporal data tables → *Not a full replacement for slowly changing dimensions, but definitely useful for auditing*

"Smart Dates" vs. "Dumb Dates" in a DW

A "dumb date" is just an attribute:

DimCustomer

CustomerSK
CustomerNK
CustomerAcquisitionDate

...

A "smart date" relates to a full-fledged Date dimension which allows significant time analysis capabilities:

DimCustomer

DimDate

FactCustomerMetrics
CustomerSK
DateSK_CustomerAcquisition

...
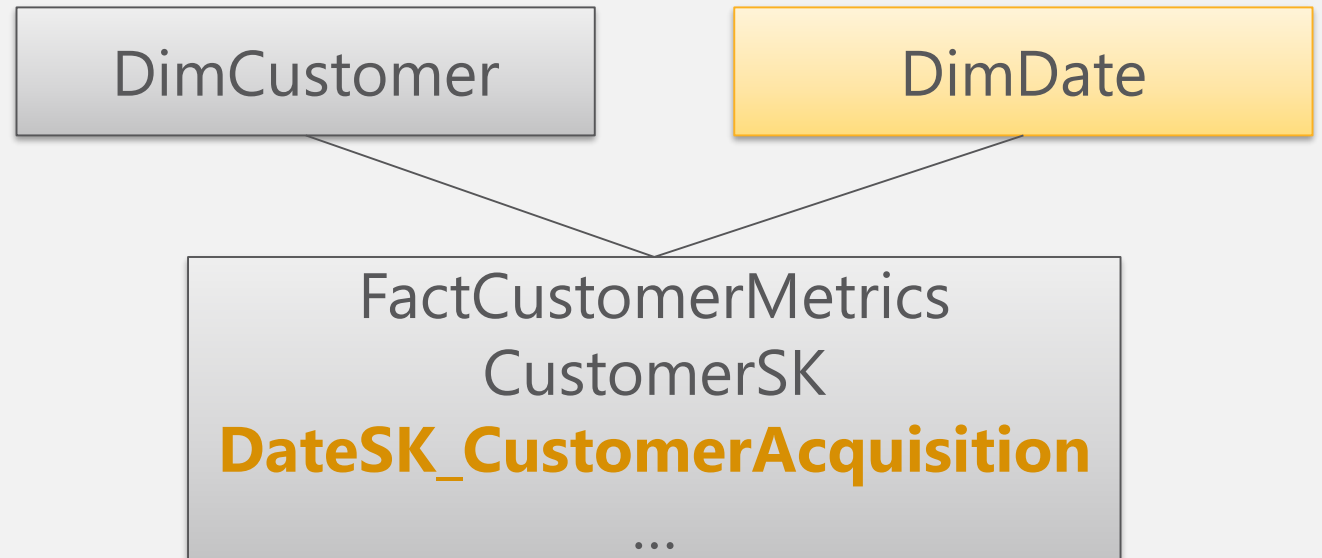
Handling of Nulls in Dimensions

Remember the NOT NULL and default constraints

Rule of thumb is to **avoid nulls** in attribute columns.

What happens with this:

SELECT CustomerType WHERE CustomerType <> 'Retail'

Too easy to forget:

SELECT CustomerType WHERE CustomerType <> 'Retail'
OR CustomerType IS NULL

Handling of Nulls in Facts

Best practice is to **avoid nulls in foreign keys**. (However, nulls are ok for a measure.)

By using an 'unknown member' relationship to the dimension, you can:
- ✓ Safely do inner joins
- ✓ Allow the fact record to be inserted & meet referential integrity
- ✓ Allow the fact record to be inserted which avoids understating measurement amounts
  Ex: Just because one key is unknown, such as an EmployeeSK for who rang up the sale, should the sale not be counted?

Views Customized for Different Purposes
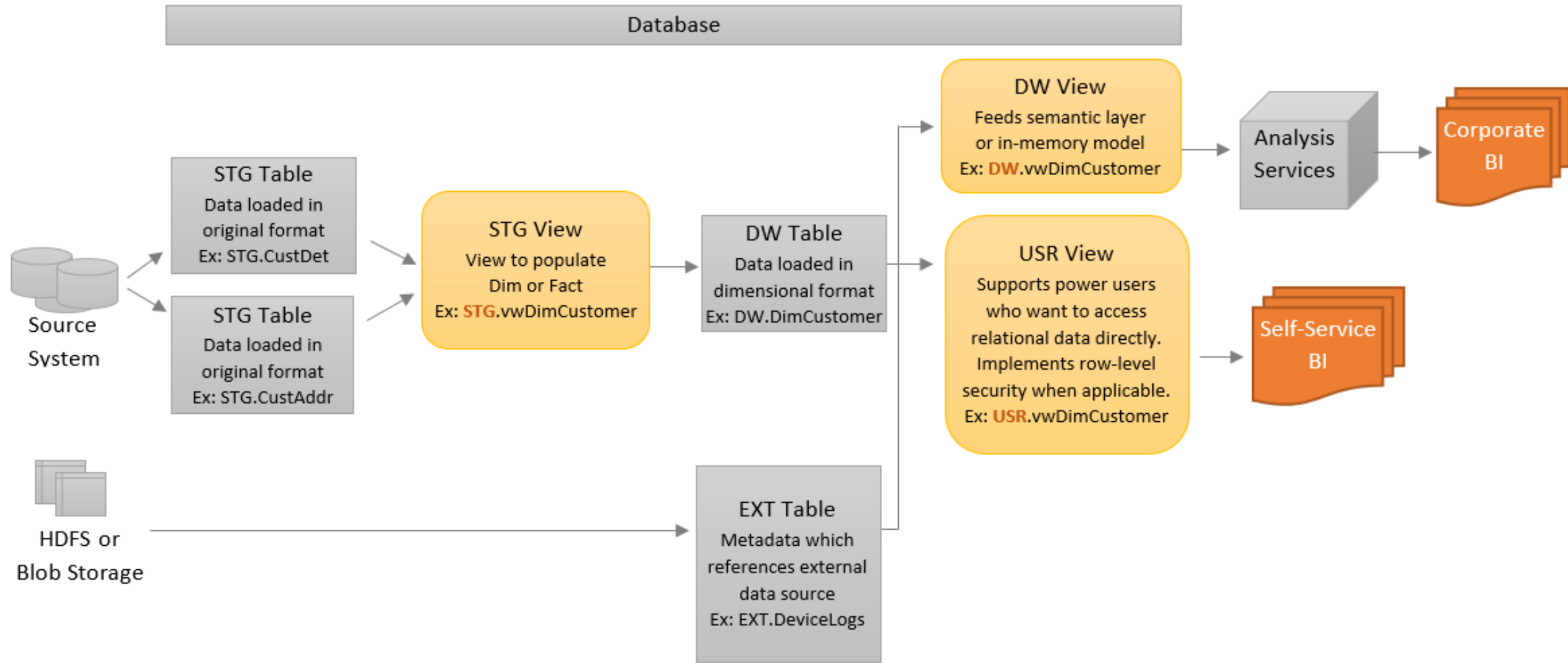
Recap of Important DW Design Principles

✓Staging as a "kitchen" area

✓Integrate data from multiple systems to increase its value

✓Denormalize the data into a star schema

✓A column exists in one and only one place in the star schema

✓Avoid snowflake design most of the time

✓Use surrogate keys which are independent from source systems

✓Use conformed dimensions

✓Know the grain of every table

✓Have a strategy for handling changes, and for storage of history

✓Store the lowest level of detail that you can

✓Use an 'unknown member' to avoid understating facts

✓Transform the data, but don't "fix" it in the DW

✓Structure your dimensional model around business processes

Recap of Important DW Design Principles

✓Design facts around a single business event
✓Always use friendly names & descriptions
✓Use an explicit date dimension in a "role-playing" way
✓Utilize bridge tables to handle many-to-many scenarios
✓Plan for complexities such as:
  ✓Header/line data
  ✓Semi-additive facts
  ✓Multiple currencies
  ✓Multiple units of measure
  ✓Alternate hierarchies and calculations per business units
  ✓Allocation of measures in a snowflake design
  ✓Reporting of what didn't occur (factless facts)
  ✓Dimensional only analysis

SSDT "Database Project" Tips

Database Project Format
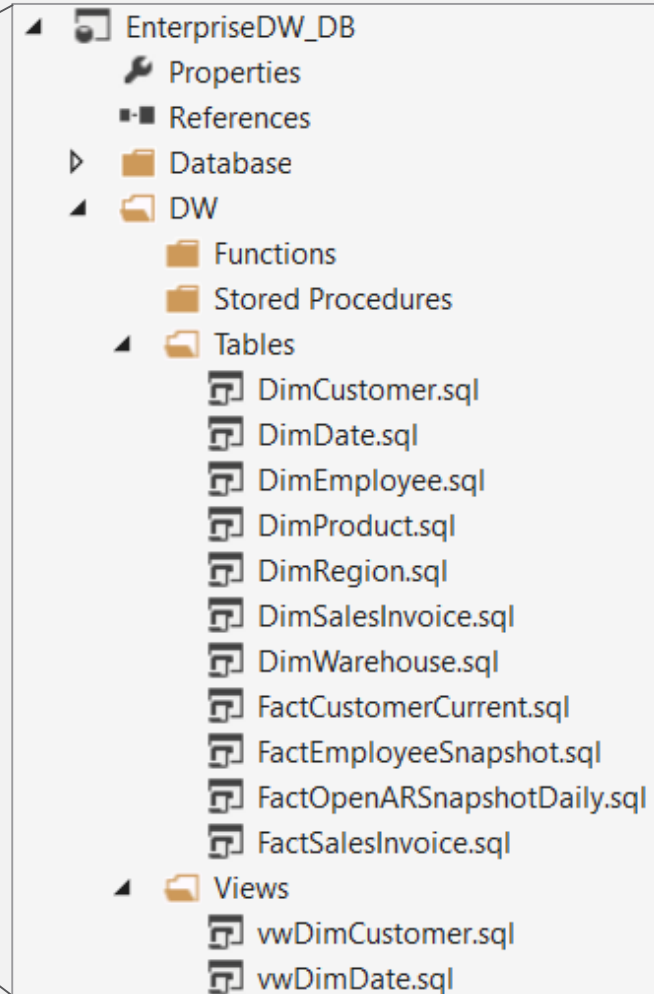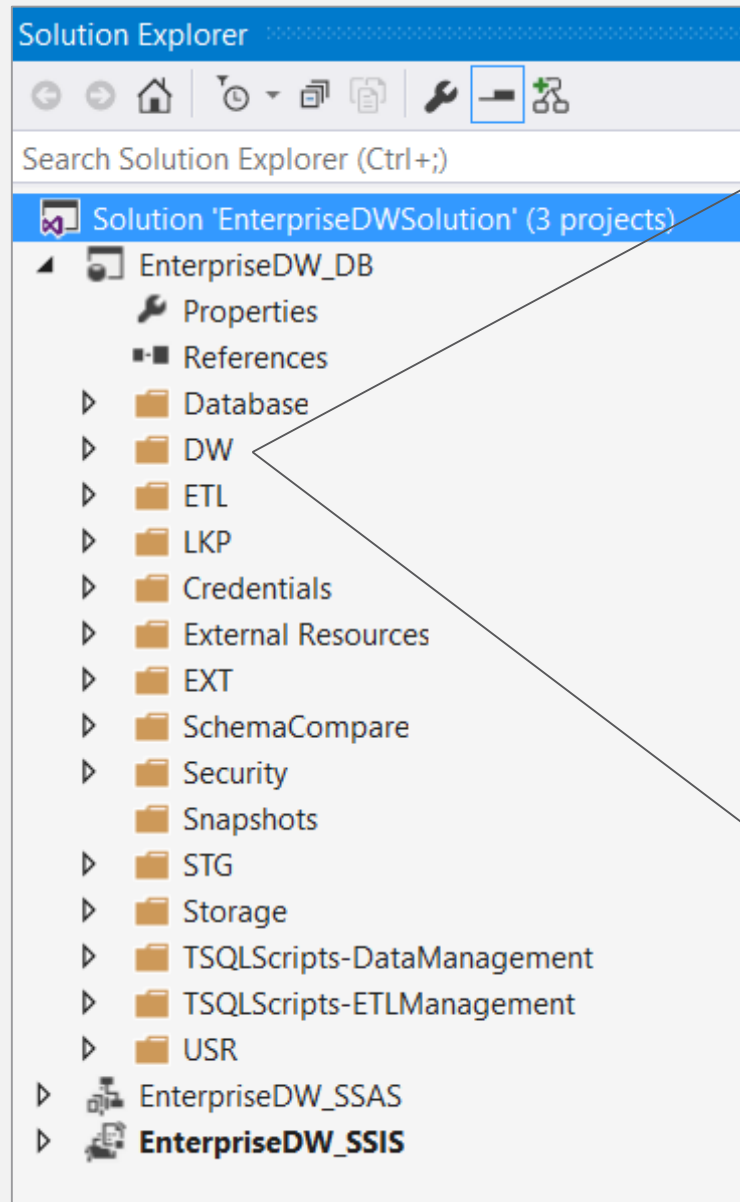
This project is organized by:
1 – Schema
   (or Category)
2 – Object Type
3 – Object

Building the Database Project

Build frequently to verify no errors or missing references

Nearly all objects should be set to Build

Database Design

```sql
1   USE [master]
2   GO
3
4 ⊟CREATE DATABASE [EnterpriseDW]
5   CONTAINMENT = NONE
6   ON  PRIMARY
7   ( NAME = N'EnterpriseDW', FILENAME = N'G:\MSSQL\Data\EnterpriseD
8     SIZE = 64MB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),
9
10  FILEGROUP [Dimensions]  DEFAULT
11  ( NAME = N'EnterpriseDW_dimensions', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_di
12    SIZE = 3GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),
13
14  FILEGROUP [Facts]
15  ( NAME = N'EnterpriseDW_facts', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_facts.ndf' ,
16    SIZE = 3GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),
17
18  FILEGROUP [Staging]
19  ( NAME = N'EnterpriseDW_staging', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_staging.ndf'
20    SIZE = 2GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB ),
21
22  FILEGROUP [Other]
23  ( NAME = N'EnterpriseDW_other', FILENAME = N'G:\MSSQL\Data\EnterpriseDW_other.ndf'
24    SIZE = 1GB , MAXSIZE = UNLIMITED, FILEGROWTH = 256MB )
25
26  LOG ON
27  ( NAME = N'EnterpriseDW_log', FILENAME = N'L:\MSSQL\Log\EnterpriseDW_log.ldf' ,
28    SIZE = 256MB , MAXSIZE = 2048GB , FILEGROWTH = 256MB )
```

Pre-sized files

Auto-grow allowed in sizeable increments (just in case)

Separate disks to locate data & log

Unknown Member Row

The SK reference in a fact table if the real value is unknown or does not exist.

> **Build action = none since this is DML**

```sql
--Step 1. Permit an explicit value to be inserted into identity column

SET IDENTITY_INSERT [DW].[DimCustomer] ON;
GO

--Step 2. Insert unknown member row

INSERT INTO [DW].[DimCustomer]
        ([CustomerSK]
        ,[RegionNumberNK]
        ,[CustomerNumberNK]
        ,[CustomerNumber]
        ,[CustomerName]
        ,[AuditRowIsCurrent])
    VALUES
        (-1
        ,N'Unknown'
        ,N'Unknown'
        ,N'Unknown'
        ,N'Unknown'
        ,'1901-01-01'
        ,'2999-12-31'
        ,1
        );
GO

--Step 3. Disable ability for an explicit value to be inserted into identity column

SET IDENTITY_INSERT [DW].[DimCustomer] OFF;
GO
```

> **Identity_Insert does require elevated permissions**

Manually Maintained Data

Maintain a DML script in a Lookup (LKP) table instead of hard-coding in the ETL.

```sql
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdateDate], [AuditUpdateBy])
VALUES ('C  ', 'Warehouse Credit', 'Warehouse', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdateDate], [AuditUpdateBy])
VALUES ('D  ', 'Direct Sale', 'Direct', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdate    ] [AuditUpdateBy])
VALUES ('R  ', 'Direct Credit', 'Direct', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUp
VALUES ('S  ', 'Reload Sale', 'Reload', GETDATE(), SUSER_SNAME() )
GO
INSERT [LKP].[SalesInvoiceOrderType] (
[OrderTypeCode], [OrderTypeDescription], [OrderTypeChannel], [AuditUpdateDate], [AuditUpdateBy])
VALUES ('T  ', 'Reload Direct', 'Reload', GETDATE(), SUSER_SNAME() )
```

> Build action = none since this is DML

Schema Compare

Schema Compare Options

Schema Compare Options

Schema Compare Options for SQL Server 2016

General | Object Types

☐ Ignore authorizer
☐ Ignore column collation
☑ **Ignore comments**
☑ Ignore cryptographic provider file path
☐ Ignore DDL trigger order
☐ Ignore DDL trigger state
☐ Ignore default schema
☐ Ignore DML trigger order
☐ Ignore DML trigger state
☑ Ignore file and log file path
☑ Ignore file size
☐ **Ignore filegroup placement**
☐ **Ignore fill factor**
☑ Ignore full text catalog file path
☐ Ignore identity seed

Specifies whether differences in the increment for an identity column should be ignored or updated when you publish to a database.

Reset

OK | Cancel

Schema Compare Options

Schema Compare Options for SQL Server 2016

General | **Object Types**

Selected object types (and their children) are included in the comparison

☑ Partition Schemes
☐ **Permissions**
☑ Primary Keys
☑ Queues
☑ Remote Service Bindings
☐ **Role Memberships**
☑ Rules
☑ Scalar-valued Functions
☑ Search Property Lists
☑ Security Policies
☑ Selective XML Indexes
☑ Sequences
☑ Services
☑ Signatures
☑ Statistics
☑ Stored Procedures
☑ Symmetric Keys
☑ Synonyms
☑ Table Type Indexes
☑ Tables
☑ Table-Valued Functions
☑ Triggers
☑ Unique Keys
☑ User-Defined Data Types
☑ User-Defined Table Types
☑ User-Defined Types (CLR)
☐ **Users**
☑ Views
☑ XML Indexes
☑ XML Schema Collections
☐ Non-Application-scoped

Project Properties

EnterpriseDW_DB   ⚲ ✕   SqlSchemaCompare1*   EnterpriseDW.sql

Project Settings

SQLCLR

SQLCLR Build

Build

SQLCMD Variables

Build Events

Debug

Reference Paths

Code Analysis

Configuration: Active (Debug)   ⌄   Platform: Active (Any CPU)   ⌄

☐ Enable Code Analysis on Build

Rules	Treat Warning as Error
☑ Microsoft.Rules.Data.Design	◼
☑ SR0001: Avoid SELECT * in stored procedures, views, and table-valued functions.	☑
☑ SR0008: Consider using SCOPE_IDENTITY instead of @@IDENTITY.	☐
☑ SR0009: Avoid using types of variable length that are size 1 or 2.	☑
☑ SR0010: Avoid using deprecated syntax when you join tables or views.	☑
☑ SR0013: Specify values for output parameters in all code paths.	☐
☑ SR0014: Maintain compatibility between data types.	☐
☑ Microsoft.Rules.Data.Naming	◼
☑ SR0011: Avoid using special characters in object names.	☑
☑ SR0012: Avoid using reserved words for type names.	☑
☑ SR0016: Avoid using sp_ as a prefix for stored procedures.	☐
☑ Microsoft.Rules.Data.Performance	☐
☑ SR0004: Avoid using columns that do not have an index as test expressions in IN predicates.	☐
☑ SR0005: Avoid using patterns that start with "%" in LIKE predicates.	☐
☑ SR0006: In the comparison, simplify the expression that includes indexed columns.	☐
☑ SR0007: Use ISNULL(column, default value) on nullable columns in expressions.	☐
☑ SR0015: Extract deterministic function calls from WHERE predicates.	☐

Schema Compare

Usually **don't** want to let the target update directly

Generates a script to use for deployment

re_Pr...ctToLocalDB.scmp × DimCustomer.sql [Design]

Compare ▮ Update ⬚ ⚙ ⟨⊟ ▾ ⊤ ⊤∅ ↑ ↓

C:\Users\mcoates\OneDrive\Visual Studio Projects\De...\EnterpriseDW_DB ⇆ SSLAP40.EnterpriseDW

Type		Source Name	Action		Target Name
˅ 📁 Delete					
⟩ 🔲	View		☑ ✖	USR.vwSensorMeasurements
⟩ 🛢	External Data Source		☑ ✖	SensorDataDS
⟩ 📄	External File Format		☑ ✖	TextFileCommaDelimitedFF
⟩ 📊	External Table		☑ ✖	EXT.SensorMeasurements
˅ 📁 Change					
⟩ ⊞	Table	DW.DimCustomer	☑ ✏	DW.DimCustomer
⟩ ⊞	Table	DW.FactARSnapshot	☑ ✏	DW.FactARSnapshot
⟩ ⊞	Table	DW.FactSalesInvoice	☑ ✏	DW.FactSalesInvoice

Data Compare
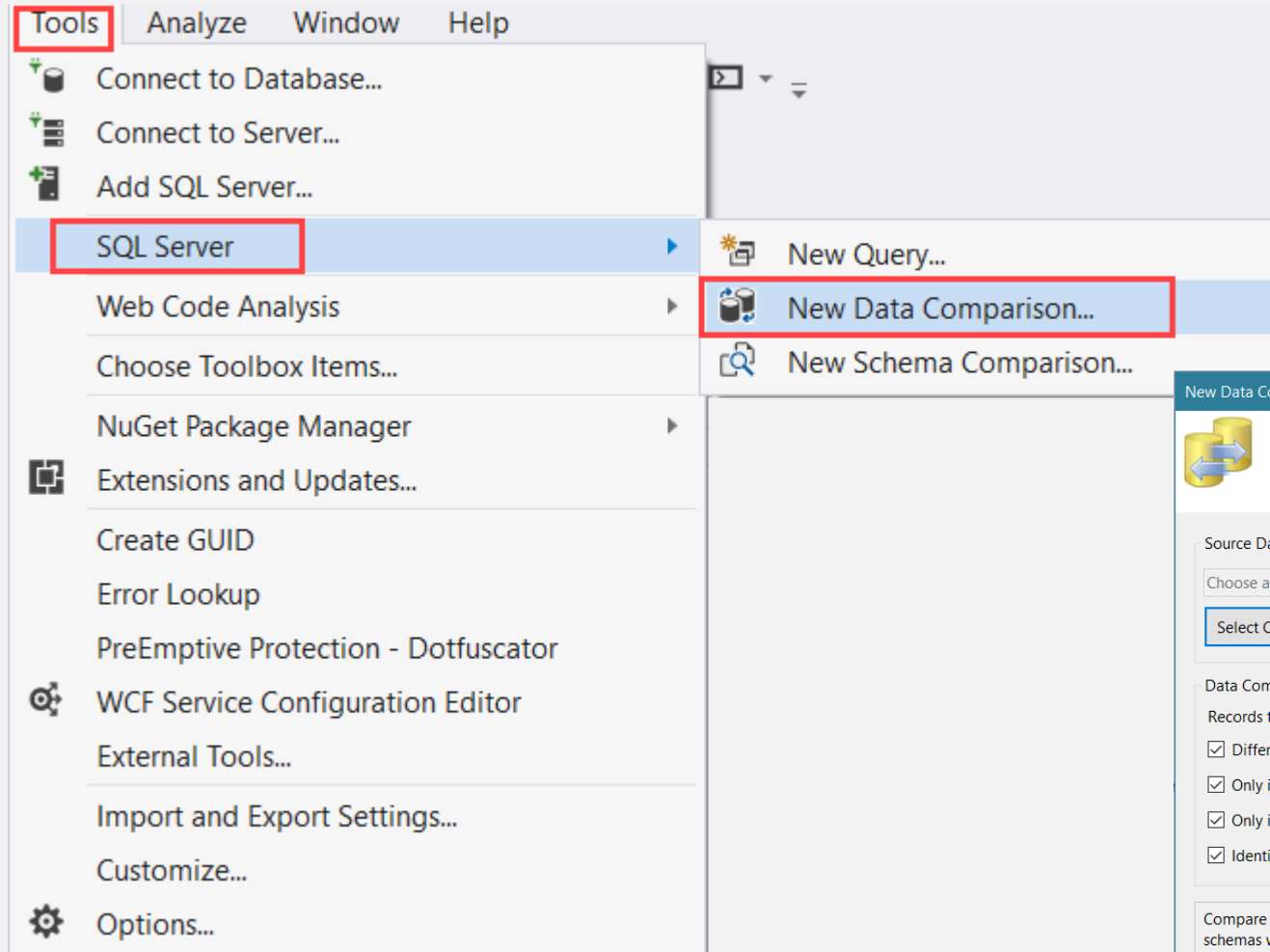
Basic functionality to compare data between two tables -- schema must match.
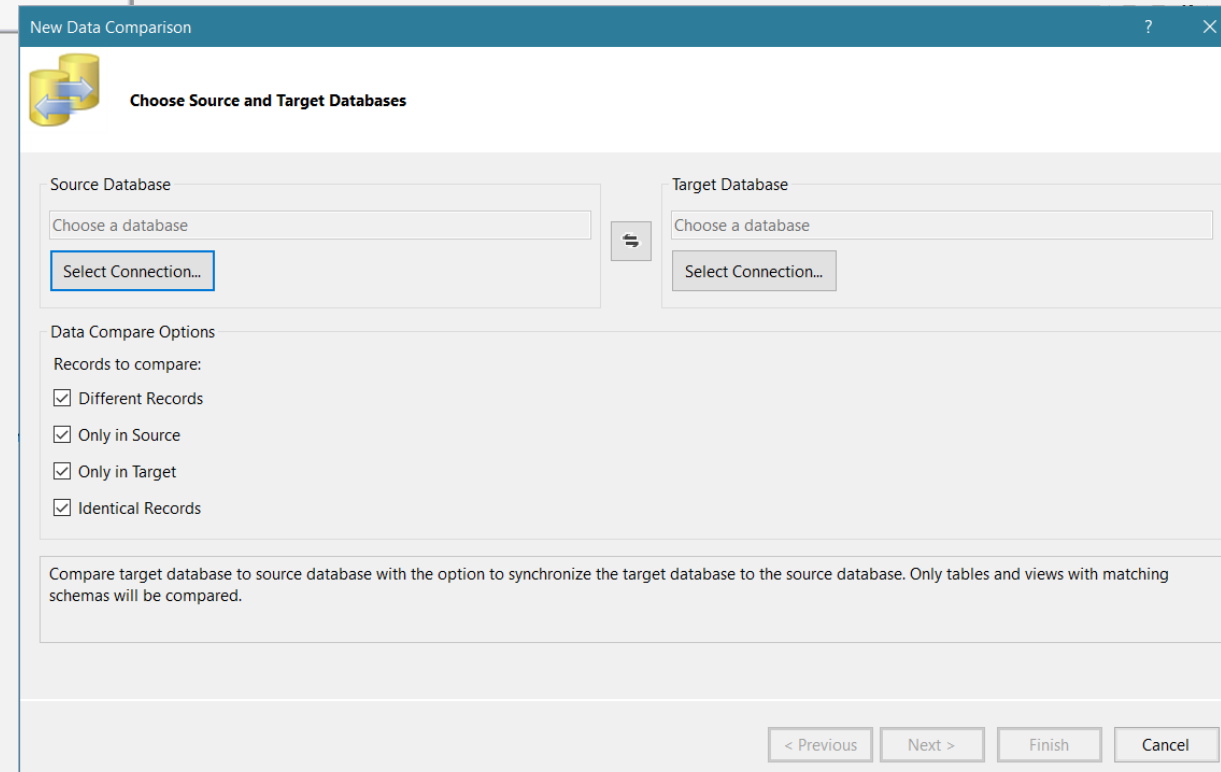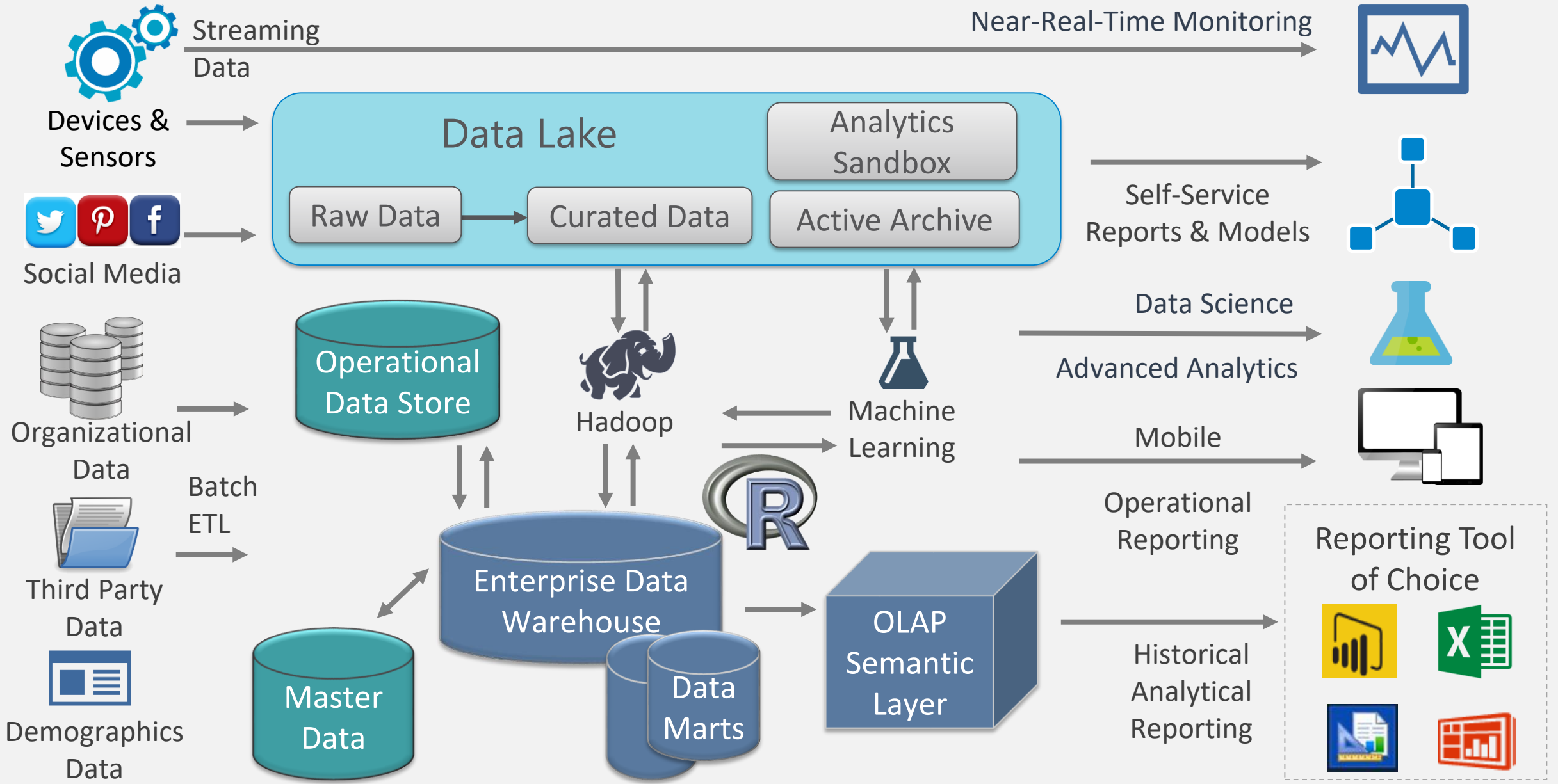
Project Snapshot

Snapshot of the database schema at a point in time (ex: major release points).

Store the .dacpac file in the project if desired
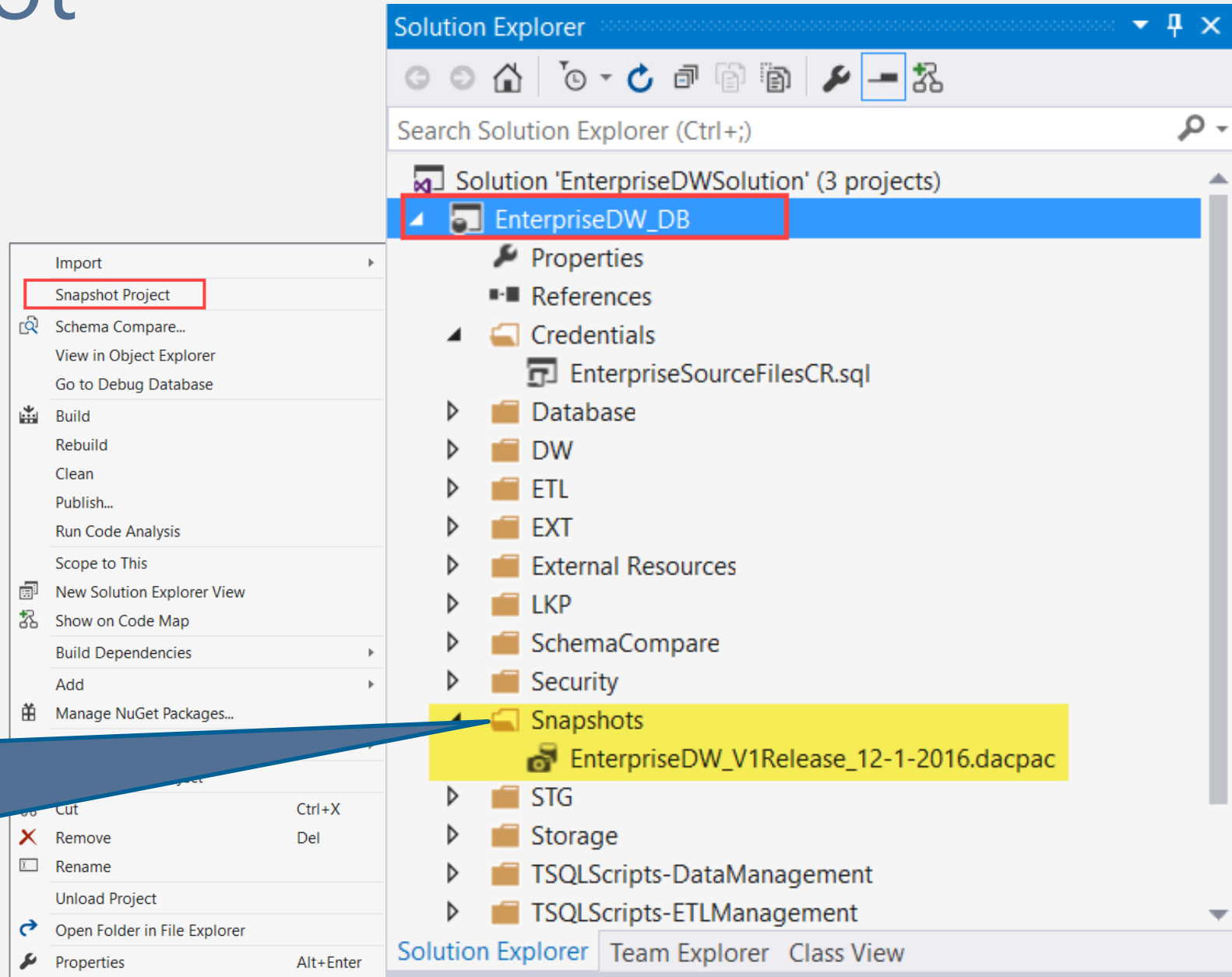
Planning Future Growth
of the Data Warehouse

Growing your DW/BI/Analytics Environment

Cloud & Hybrid Platforms

Real-Time Reporting

Modern DW Multi-Platform Architecture

Self-Service BI

Advanced Analytics

Agile, Nimble Solutions

Achieving Extensibility in a DW

Design with change in mind. Ex: Create a lookup table with code/descriptions, or implement in a view, rather than hard-coding in ETL.

Plan for a **hybrid** environment with **multiple architectures**.

Introduce **conformed dimensions first** whenever possible.

Try to **avoid isolated "stovepipe" implementations** unless the isolation is absolutely intended.

Conduct **active prototyping sessions** with business users to flush out requirements. A data modeling tool like Power BI works well for this.

Achieving Extensibility in a DW

Be prepared to do some **refactoring** along the way. Ex: converting an attribute to be a conformed dimension.

First implementation:

DimCustomer

CustomerName
CustomerRegion

...

FactSalesInvoice
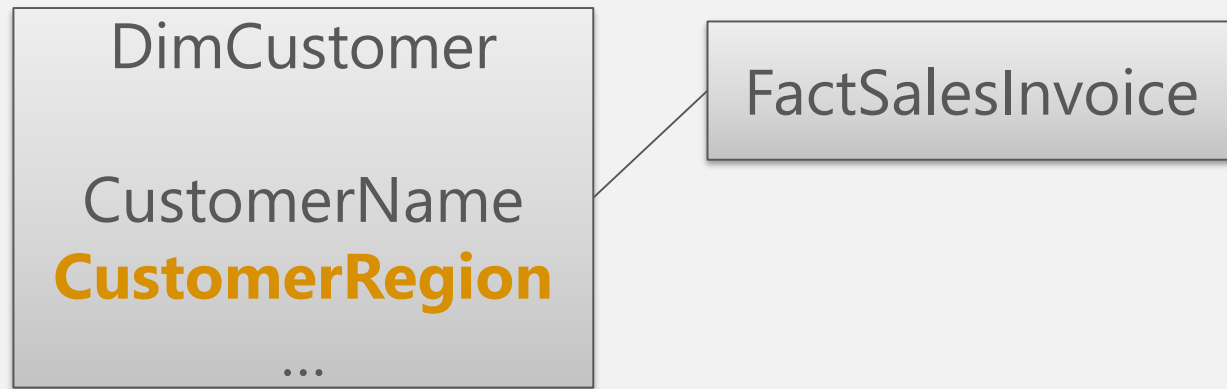
Updated in a later iteration:

FactWarrantyRequest

DimRegion

DimCustomer

FactSalesInvoice
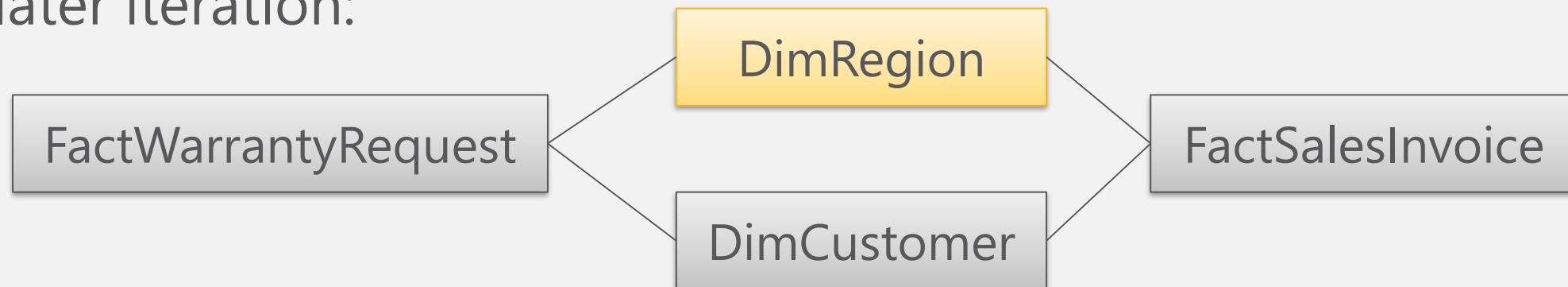
Achieving Extensibility in a DW

Introducing new measures:
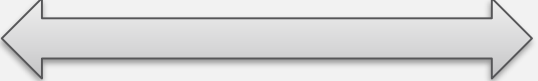- Can be a new column in a fact table as long as it's the same grain & the same business process

Introducing new attributes:
- Can be a new column in a dimension, or
- Can be via a new foreign key in a fact table as long as it doesn't affect the grain

Agility for the things that usually require the most time investment:
- Data modeling
- ETL processes
- Data quality

Achieving Extensibility in a DW

Reusability Downstream ⟷ Speed of Change Implemented
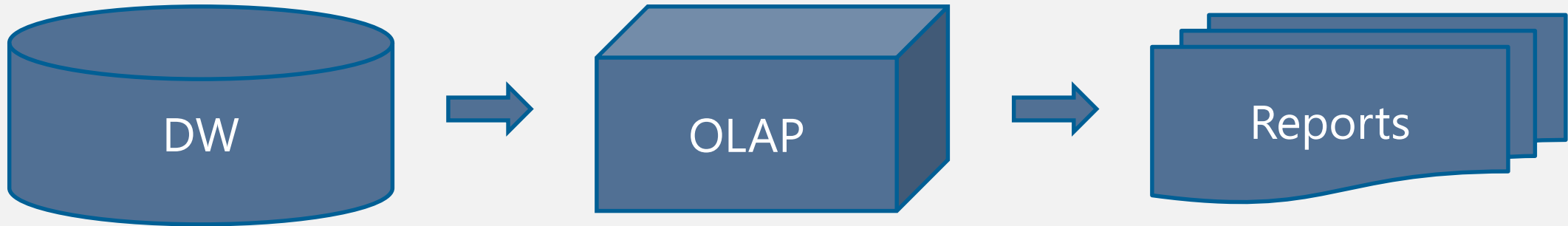
Consider using an ***OLAP cube or in-memory model*** (like Analysis Services) for:

- Summary data (as opposed to summary tables in your DW)
- Year-to-Date type of calculations
- Year-over-Year type of calculations
- Aggregate level calculations (as opposed to row-by-row calculations)

Modern DW: Important Concepts to Know

| Polygot Persistence	Lambda Architecture	Schema on Read
Using the most effective data storage technology to handle different data storage needs	Data processing architecture which supports large amounts of data via a speed layer, batch layer, and serving layer	Data structure is applied at query time rather than when the data is initially stored

Recommended Resources
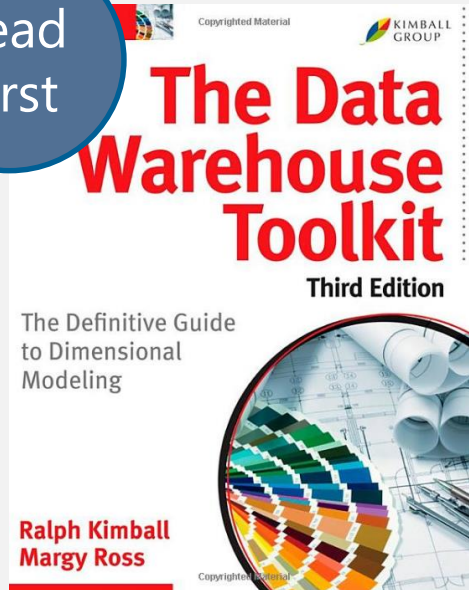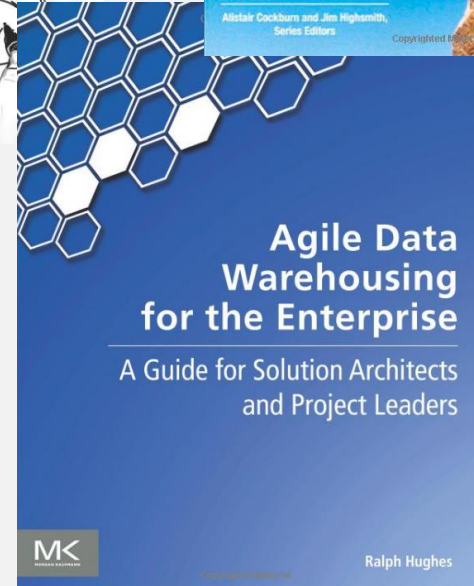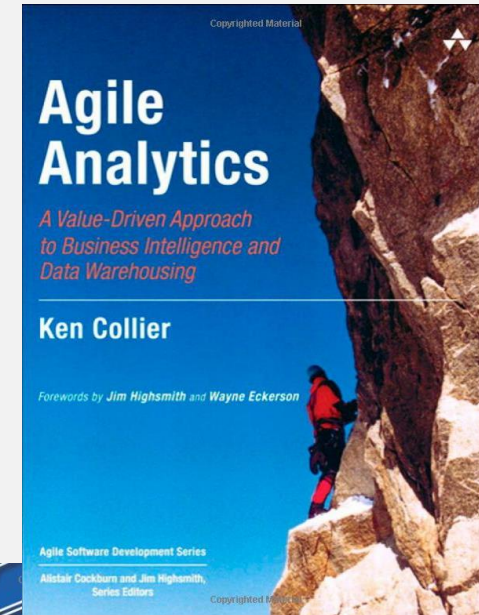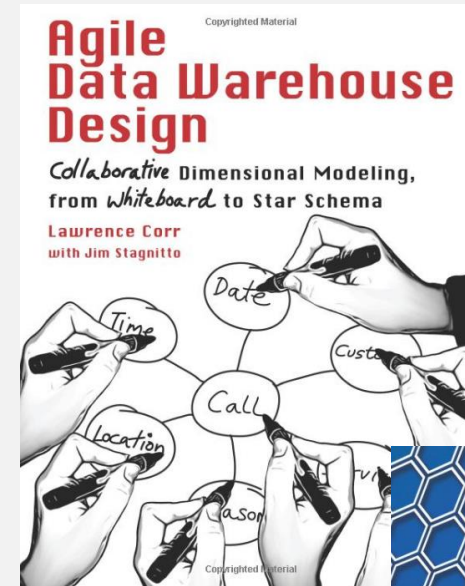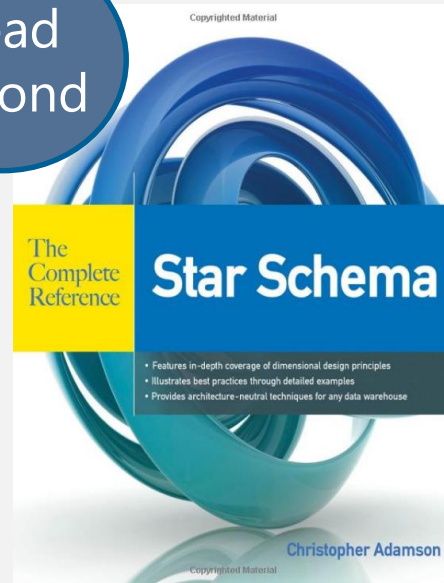
Thank You for Attending

To download a copy of this presentation:

SQLChick.com "Presentations & Downloads" page
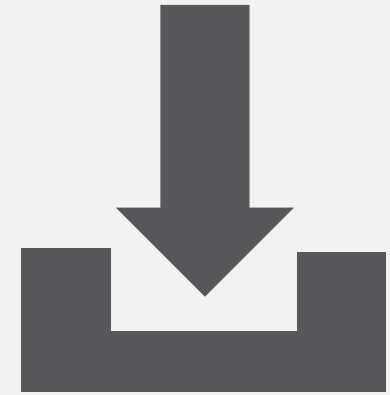
Melissa Coates
BI Architect, SentryOne
sentryone.com

Blog: sqlchick.com
Twitter: @sqlchick